**Camera Transforms in Computer Vision**

An Exploration of the Mathematics and Implementation of Computer Vison Coordinate Systems

Ricardo A. Carrillo

University of Washington Bothell

School of STEM

Applied Computing Capstone

Table of Contents

[Introduction 3](#_Toc516243814)

[What is a Camera Transform? 4](#_Toc516243815)

[How to use this paper 4](#_Toc516243816)

[Section I: Coordinate Systems Defined 5](#_Toc516243817)

[GPS 5](#_Toc516243818)

[GPS Coordinate Pseudo code 6](#_Toc516243819)

[GPS Coordinate Implementation Concerns 6](#_Toc516243820)

[Cartesian 7](#_Toc516243821)

[Right handed 7](#_Toc516243822)

[Left handed 7](#_Toc516243823)

[World Centered Cartesian 8](#_Toc516243824)

[World Coordinate Pseudo code 9](#_Toc516243825)

[World Coordinate Implementation Concerns 9](#_Toc516243826)

[Camera Centered Cartesian 10](#_Toc516243827)

[Camera Coordinate Pseudo code 11](#_Toc516243829)

[Camera Coordinate Implementation Concerns 11](#_Toc516243830)

[Camera Pixel Coordinates 12](#_Toc516243831)

[Camera Pixel Pseudo code 13](#_Toc516243833)

[Camera Pixel Implementation Concerns 13](#_Toc516243834)

[Section II: The GPS to World Centered Transform 14](#_Toc516243835)

[Haversine Formula 15](#_Toc516243836)

[Moving the Origin 16](#_Toc516243837)

[1. Rotate the coordinate system. 16](#_Toc516243838)

[2. Translate the coordinate system. 17](#_Toc516243839)

[Reversal 17](#_Toc516243840)

[1. Find the world coordinate of the new origin, revert to earth center. 17](#_Toc516243841)

[2. Revert rotation of the coordinate system to align with earth center. 17](#_Toc516243842)

[3. Reproject to spherical space 17](#_Toc516243843)

[GPS to World Centered Pseudo code 18](#_Toc516243844)

[GPS to World Implementation Concerns 19](#_Toc516243845)

[Section III: The World Centered to Camera Centered Transform 20](#_Toc516243846)

[Rotation Matrices 20](#_Toc516243848)

[Inverse Matrices 22](#_Toc516243849)

[Homogenous Coordinates 23](#_Toc516243850)

[Conversion to Homogenous Coordinates 23](#_Toc516243851)

[Advantages of Homogenous Coordinates 24](#_Toc516243852)

[Disadvantages of Homogenous Coordinates 24](#_Toc516243853)

[Translation Matrices 24](#_Toc516243854)

[Appending Translation Matrices 25](#_Toc516243855)

[Camera Matrix 26](#_Toc516243856)

[Reversal 26](#_Toc516243857)

[Expanded Camera Matrix (Camera Centered to World Centered Transform) 26](#_Toc516243858)

[Expanded Inverse Camera Matrix (World Centered to Camera Centered Transform) 26](#_Toc516243859)

[World Centered to Camera Centered Pseudo code 27](#_Toc516243860)

[Camera Centered to World Centered Pseudo code 27](#_Toc516243861)

[Implementation concerns 27](#_Toc516243862)

[Section IV: Camera to Camera Plane Projection 28](#_Toc516243863)

[Pin hole Camera Model 28](#_Toc516243864)

[Projection of point on to a plane (perfect camera) 28](#_Toc516243865)

[Camera Plane Projection Pseudocode 30](#_Toc516243866)

[Implementation Concerns 30](#_Toc516243867)

[Issues encountered and continuing research 31](#_Toc516243868)

[Camera Aberration & Distortion 31](#_Toc516243869)

[Reprojection and the distance problem 31](#_Toc516243871)

[Software Validation 32](#_Toc516243872)

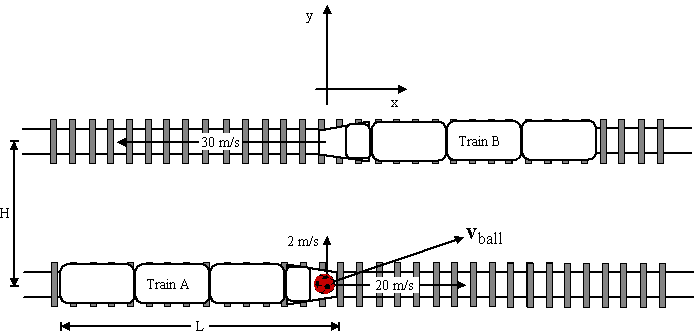
[Bibliography 32](#_Toc516243873)

[Acknowledgments 32](#_Toc516243874)

# Introduction

In mathematics, physics, computer graphics, computer vision, and many other disciplines, it is often advantageous to define coordinate systems that make a problem simpler to understand and easier to do. Being able to pick and choose a coordinate system that makes calculations easier is one of the most basic of techniques in these field of study.

Take the following example, in the case of a famous physics problem in which a soccer ball is kicked off the back of a moving train, from the perspective of an observer on the train, the ball will be kicked away at a speed of 20 m/s. From the point of view of an observer on the ground, the ball may move much more slowly or even have zero velocity when kicked off the train. Which observer’s observation of the ball’s speed is correct? In this case, both are, but the correct answer depends on the coordinate system one chooses.



**Figure 1.** Example of simple coordinate transform

In this case, moving from the train coordinate system to the earth coordinate system involves adding or subtracting the velocity vectors of the ball in one or two dimensions. The speed of the ball will simply be the addition of the speed imparted on it by the kicker, and the movement of the train. Having the ability to look at the same scene from different coordinate systems allows the problem to be solved with ease, gaining deeper insight by viewing the same systems in different ways. Therefore, it is useful to define, change, and *transform* between coordinate systems. This example is called a Galilean transform and is among the simplest of coordinate transforms.

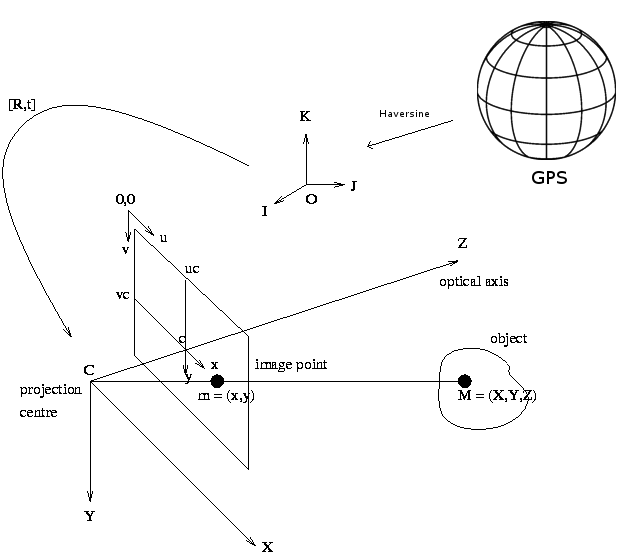
But what happens if one wants to define a system that involves multiple moving objects in three dimensions? What about when one coordinate system does not have a flat geometry (such as what would be found on a globe), an d what if the system needs to be compatible with systems that use flat cartesian coordinate systems? Spatial information must be accurate, mathematically consistent, and agree between multiple differently defined coordinate systems. Therefore, there needs to be a mathematical expression or system that can *transform* between coordinate systems. This is the definition of a coordinate transform. The design of such a system for use with computer vision navigation systems is the focus of this paper.

## What is a Camera Transform?

Machine vision is a technology in which electronic imaging is used to carry out navigation, telemetry information, spatial processing, or any other computerized task that depends on visual data. Because these systems require the integration of several data streams to function, each is defined in a different coordinate system. The ability to jump between these coordinate systems is required. A Camera Transform is the collection of mathematical procedures needed to handle these jumps between coordinate systems.

## How to use this paper

The Camera Transform outlined in this paper uses the following coordinate systems:

* GPS
* World Centered Cartesian
* Camera Centered Cartesian
* Camera Pixel Coordinate Position

The overall theoretical layout is as follows:

**Figure 2.** Coordinate Systems and Transforms Visualized

Ultimately, the purpose of this paper is to define the mathematical framework that any self-driving vehicle could use for the purposes of navigation, define the coordinate systems used for this purpose, specify how to transform between them, and define a mock interface for a software implementation.

# Section I: Coordinate Systems Defined

## GPS

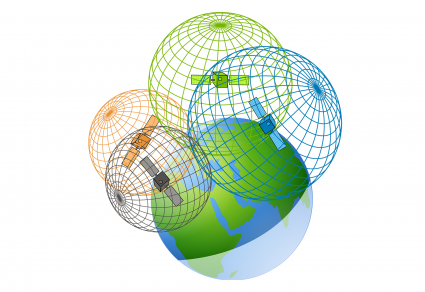
The Global Positioning System (GPS) is a navigational system developed by the US military which provides location and time information to any GPS receiver in the world. GPS works through a network of satellites that orbit earth, each carrying incredibly precise atomic clocks. A GPS receiver receives signals from these satellites and can triangulate a position by measuring the delays in the signal received from each satellite. This allows the GPS receiver to calculate its absolute latitude longitude and elevation on the earth to an accuracy of within 4 meters (military spec). While not accurate enough for the purposes of vision systems, it is a starting point for telemetry.

A GPS coordinate contains the following components:

**Degrees Latitude:** Measures displacement on the N/S axis of the globe from the equator. Measured in degrees, minutes, seconds.

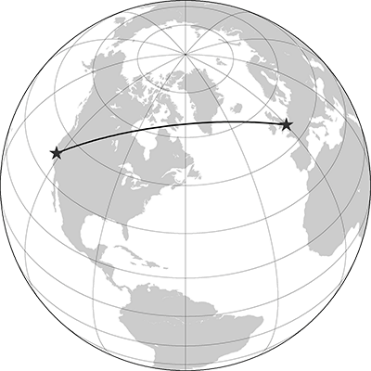
**Degrees Longitude:** Measures displacement on the W/E axis of the globe from the prime meridian. Measured in degrees, minutes, seconds.

**Elevation:** Measure of altitude. Measured in meters above or below sea level

It is important to note that GPS is not a flat coordinate system geometrically. A “straight” line on a globe follows a curved “great circle”. One degree of latitude is not the same as one degree of longitude, and the distance between each degree changes with displacement north or south.

**Los Angeles USA:** [*34°03′N, 118°15′W*](https://tools.wmflabs.org/geohack/geohack.php?pagename=Los_Angeles&params=34_03_N_118_15_W_region:US-CA_type:city(3792621))*, Ele: 93m*

**Berlin Germany:** *52°31′00″N, 13°23′20″E, Ele: 34m*

****

**Figure 3.** GPS Coordinates Visualized

### GPS Coordinate Pseudo code

Struct GPSCoordinate //Represents a GPS coordinate

{

Float lat, long, ele;

String latDir, longDir //Direction away from equator and prime meridian.

GPSCoordinate()//Defaults at origin 0,0,0

GPSCoordiante(Lat,LatDir, Long, LongDir, Ele)

}

### GPS Coordinate Implementation Concerns

* GPS Coordinates can never be negative. Inputs of negative GPS inputs instead change the accompanying string to either N/S or W/E depending on direction of rotation. (-20W = 20E)
* If input elevation is less than the radius of the earth, GPS points must be remapped to a point on the opposite end of the sphere. (Elevation of -12743Km = Elevation 1km on opposite side of planet). This application is not suitable for purposes involving deep points inside the earth.
* GPS Coordinates operate on modulus 360 degrees.

## Cartesian

A three-dimensional cartesian coordinate system is a coordinate system that specifies a location in flat three-dimensional space as a triplet of coordinates, each containing the component of the location measured along one of three orthogonal number line, or axes. It can also be defined as the orthogonal intersection of three planes. Cartesian coordinate systems may be constructed in either a left, or right-handed orientation. It is essential to know how the orientation is defined in a machine vision coordinate system, as axes orientations define the behavior of a translation or rotation. Signs must be flipped depending on the defined axis directions.

### Right handed

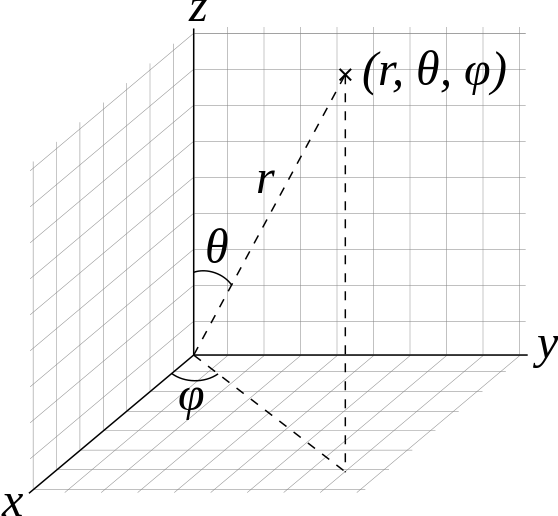
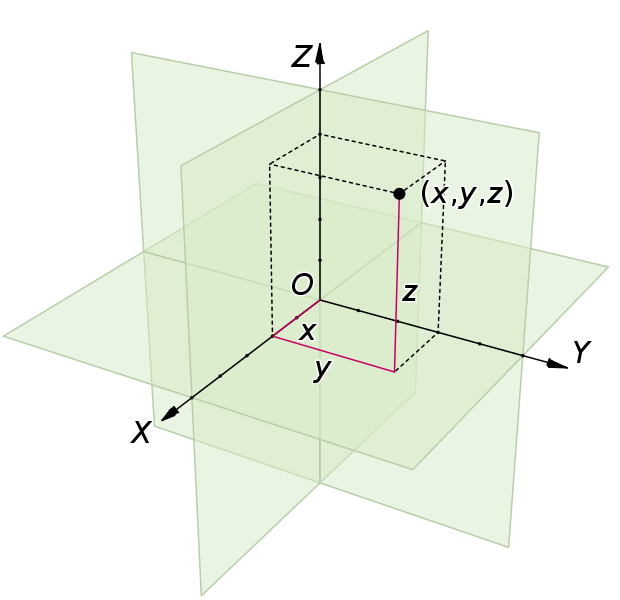
If the XY plane is horizontal, the Z axis points up.

### Left handed

If the XY Plane is horizontal, the Z axis points down.

**Figure 4.** Left/Right handedness of cartesian coordinates

Cartesian Coordinate systems may be defined rectangularly (X,Y,Z) or spherically. The spherical definition is comprised of a radial distance r, with two angles *θ*, *φ*, where *θ* (theta) is the polar angle and *φ* is the azimuthal angle. The two are related by the trigonomic identities. Both representations are used later.



**Figure 5.** Spherical and rectangular cartesian coordinates

## World Centered Cartesian

The world coordinate system is the first cartesian coordinate system transformed to after GPS. Since on small scales the curvature of the earth is negligible, it would be correct to approximate the world as a flat horizontal plane extending infinitely in all directions. The World Centered Cartesian system (or *world coordinates* for short) which is oriented with the cardinal directions, is defined as such:

Axis Orientation: Left Handed.

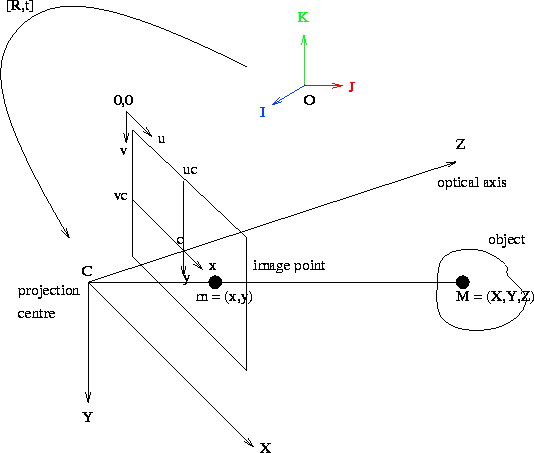
On = (Oi,Oj,Ok) : An arbitrary point on the world coordinate system.

O0 = (0,0,0) : The origin of the World Coordinate system. Can be defined as any arbitrary point on the earth’s surface. Note, that although this can be any point on the earth, the GPS transform must be calibrated to this origin coordinate.

Oi : The i direction (X axis) in line with the North/South direction. Defined in meters. Positive numbers are meters north of the origin. Negative numbers are south of origin.

Oj: The j direction (Y axis) in line with the West/East cardinal directions. Defined in meters. Positive numbers are meters West of origin. Negative numbers are East of origin.

OK : The k direction (Z axis) in in line with elevation. Defined in meters above sea level. Positive numbers are above sea level. Negative numbers are below sea level.



**Figure 6.** World Coordinates Highlighted

### World Coordinate Pseudo code

Struct WorldTriple //Represents a World coordinate

{Float Oi,Oj,Ok;

WorldTriple()//Defaults at origin 0,0,0

WorldTriple(float Oi, float Oj, float z=Ok)

}

### World Coordinate Implementation Concerns

* Different 3D graphics API may define coordinate systems differently. Coordinate system orientation must be checked with arguments and signs switched as necessary.
* When calculating distances, the absolute value of any coordinate additions must be taken to avoid distances in the complex plane. (sqrt(-1))
* Assumes earth is a perfect sphere
* Floating point error may occur for very small angles such as cos(0.9999999) = cos(1) which is false. More accurate data types must be used when working on very small scales.

## Camera Centered Cartesian

The self-driving vehicle must have its own coordinate system rotated in line with its orientation. The world coordinate system rotated and translate to match the orientation of the vehicle’s camera is called the Camera Centered Cartesian (Camera Coordinate). The Camera Coordinate System is defined as such:

Axis Orientation: Right Handed.

Mn = (Cx,Cy,Cz) : An arbitrary point in the Camera Coordinate system

M0 = (0,0,0) : The origin of the Camera Coordinate System. Centered on the lens of the autonomous vehicle, this specifies the current location of the vehicle. Corresponds with the vehicle’s displacement from the origin of the World Coordinate system.

Cx: The X axis extends to the right of the vehicle. Defined in meters, positive numbers are to the right of the vehicle, negative are to the left.

CY: The Y axis extends downwards from the lens. Defined in meters. Positive numbers are below the lens. Negative numbers are above the lens.

CZ : The Z axis extends out from the front of the camera lens. Defined in meters, positive numbers are in front of the lens. Negative numbers are behind the lens. Also called the optical axis.

## 

**Figure 7.** Camera Centered Coordinates Highlighted

### Camera Coordinate Pseudo code

Struct CameraTriple //Represents a Camera coordinate

{

Float Cx,Cy,Cz;

CameraTriple() //Defaults at origin 0,0,0

CameraTriple(float Cx, float Cy, float Cz)

}

### Camera Coordinate Implementation Concerns

* The Camera Centered Cartesian Coordinate System assumes the camera is at the spatial center of the vehicle looking in the Z direction, displacement of the vehicle is equal to displacement of the camera. The centering of the coordinate system will have to be corrected depending on how the camera is placed on the vehicle.
* Different 3D graphics API may define coordinate system axis differently. Coordinate system orientation must be checked with arguments and signs switched as necessary.
* Floating point error may occur for very small angles such as cos(0.9999999) = cos(1) which is false. More accurate data types must be used when working on small scales.

## Camera Pixel Coordinates

Camera Pixel Coordinates are the location of an object in the camera viewport after it has been projected from camera coordinates onto the camera pixel plane. Camera pixel coordinates are defined as such:

mn = (x,y) : An arbitrary point on the camera screen. Defined in pixels. The number of pixels on the screen depend on the image resolution. Conversion to metric measurements depend on pixel size, focal length, distortion coefficients, and resolution.

m0,0 = (0,0): The origin of the Camera Pixel Coordinate system. Defined as the top left point on the camera image. Units are in pixels.

mx : Displacement on the horizontal axis of the camera pixel plane. If the image has a resolution of 64x64, 0 is the left most point on the image. 63 is the right most. Units are in pixels.

my : Displacement on the vertical axis of the camera pixel plane. If the image has a resolution of 64x64, 0 is the top most point on the image. 63 is the bottom most. Units are in pixels.

# 

**Figure 8.** Camera Pixel Coordinates Highlighted

### Camera Pixel Pseudo code

Struct CameraPixel //Represents a Camera Pixel coordinate

{

Float mx,my;

CameraPixel(res)//Defaults at origin 0,0

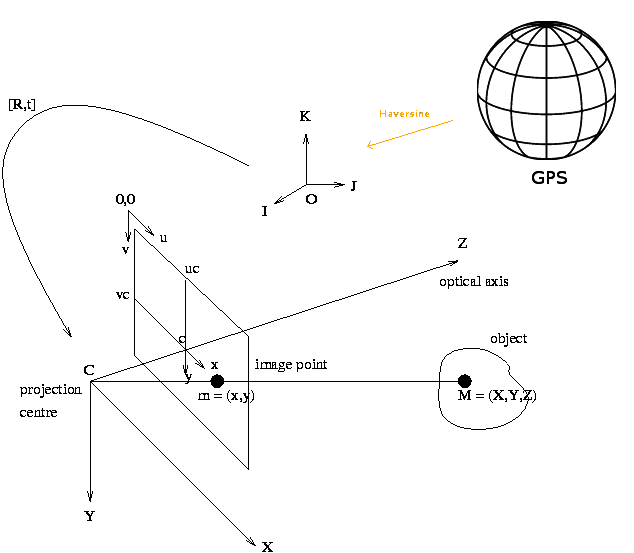
CameraPixel(float mx, float my, int res)

}

### Camera Pixel Implementation Concerns

* Camera Pixels cannot be negative valued or exceed the resolution of the camera censor. Values exceeding these ranges are outside the camera plane, and not visible on the image.
* Metric length represented by one pixel depends on camera characteristics. Camera scaling should be based on experimental results with calibration objects (chessboard)
* Different 3D graphics API may set coordinate systems differently. Check coordinate system orientations and switch arguments and signs as necessary.

# Section II: The GPS to World Centered Transform

Now that the coordinate systems to be used in the camera transform are defined, the first coordinate transform in the sequence needs to be carried out. The first task is converting from the GPS coordinate system to the World Coordinate system using the law of Haversines.

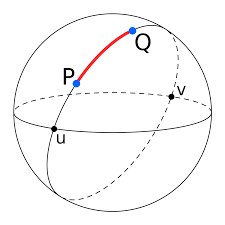
**Figure 9.** Haversine is the transition between GPS and World Coordinates

The purpose of the haversine transform is the “flattening” of the coordinate system from the spherical (or ellipsoidal) GPS system, to a flat cartesian system centered at the center of the earth. ­­­Such a change in the geometrical mapping of the coordinate system is required since the camera systems use flat cartesian coordinates for all calculations involving the optical systems, but telemetry is received and ultimately reported in GPS coordinates. GPS marks the beginning and end points of the coordinate transform.

## Haversine Formula

The Haversine Formula determines the great-circle distance between two points on a sphere in a “straight line” (geodesic) along the surface of a sphere or ellipse. Through any two points that are not directly opposite each other there is a unique great circle distance. Taking advantage of this fact, the surface of the sphere is stretched onto a 2d surface through the haversine transform. The law of Haversines can be used to carry out this geometric conversion.

The Haversine Formula is stated as follows:



(1)

Where

hav is the haversine function:

(2){\displaystyle \operatorname {hav} (\theta )=\sin ^{2}\left({\frac {\theta }{2}}\right)={\frac {1-\cos(\theta )}{2}}}

* *d* is the distance between the two points along the geodesic

**Figure 10.** Great Circle Distance

* *r* is the radius of the sphere (earth)
* *φ*1, *φ*2: latitude of point P and latitude of point Q, in radians
* *λ*1, *λ*2: longitude of point P and longitude of point Q, in radians

Solving for *d* by taking the inverse of the haversine function, gives:



Which can be explicitly expanded to:



(3) *d*

Substituting *d* in equation (3) with the distance formula in three dimensions



(4) *d* =

setting X1, Y1, Z1 to the origin at (0,0,0) and then solving for X2, Y2, Z2, gives:

(5)

(6)

(7)

Which is a coordinate triplet in the world coordinate system.

By solving equations (4), (5), & (6) for the latitude, longitude, and elevations, the reversal of the haversine formula can easily be found. Equations (7) and (8) give the latitude and longitude of any point known in world coordinates.

(8)

(9)

where asin and atan are the inverse trig functions in two dimensions.

These equations collectively give a direct mapping of the spherical *GPS System* to the flat Euclidean *World Centered Cartesian System* with an origin at the center of the earth. From this step, it would be a simple translation and rotation of the coordinate system to set any arbitrary point on earth’s surface as the origin of the world coordinate system.

## Moving the Origin

A self-driving vehicle may find it easier to set the origin of the world coordinate system at any point on earth’s surface that is convenient; rather than the center of the planet. Moving the origin is a simple matter of translating and rotating the entire world coordinate system, and is done through the following steps:

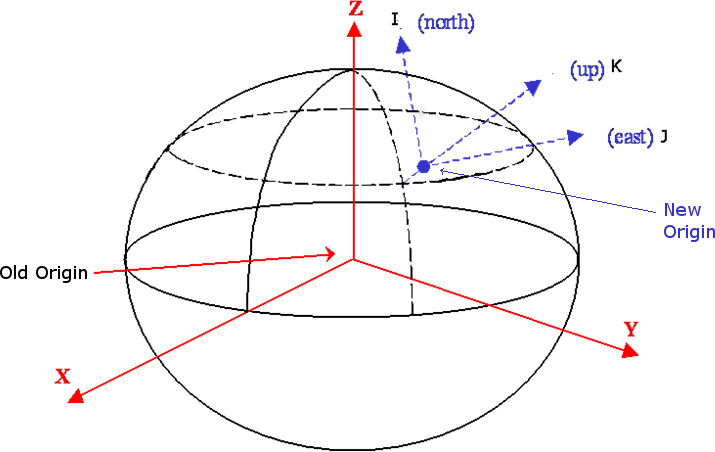
### 1. Rotate the coordinate system.

The rotation is simply a rotation in two directions, given by the latitude and the longitude. Rotate every point on the coordinate system by the latitude and longitude which correspond to the polar and azimuthal angle of the system. The world coordinate system is now aligned in the direction of the new origin.

### 2. Translate the coordinate system.

The translations will be given by inputting the latitude and longitude of the new world coordinate system origin in to equations (8) and (9).

Once the required translation is found, add the result to every point on the coordinate system. The world coordinate system’s origin is now the selected point on earth’s surface instead of the core, with the axis aligned with the cardinal directions.



**Figure 11.** World Coordinate Origin set to earth’s surface

## Reversal

The reversal of the GPS to World Coordinate transform is done by simply undoing the previous steps:

### 1. Find the world coordinate of the new origin, revert to earth center.

Use equations (8) and (9) to find the world coordinates of the origin if the system was centered at earth’s core instead. Subtract these coordinates from every point on the system to center the world coordinate system back to the center of the earth.

### 2. Revert rotation of the coordinate system to align with earth center.

Rotate the world coordinate system by the negative of the latitude and longitude. The world coordinate system is configured as illustrated in figure TBA.

### 3. Reprojection to spherical space

Use equations (8) and (9) again to solve for the GPS coordinate of any point in the world coordinate system. The transformation has been reversed.

## GPS to World Centered Pseudo code

WorldTriple GPSToWorld (GPSCoordinate input, WordlTriple origin) //Converts GPS to World

{

Float r = 6356.7; //average radius of earth (does not take elevation changes into account)

float Oi,Oj,Ok;

Oi = r \* cos(input,lat) \* cos(input.lon); //x

Oj = r \* cos(input.lat) \* sin(input.lon); //y

Ok = r \* sin(input.lat); //z, Assumes sea level

WorldTriple res = WorldTriple moveOrigin(x,y,z); //Rotates and Translates result to new origin

Return res;

}

GPSCoordinate WorldToGPS (WorldTriple point)// Converts World back to GPS

{2` m

Float r = 6356.7;

Float lat, long;

WorldTriple recenter = WorldTriple recenterOrigin(point); //Recenters Origin

lat = asin(recenter.z / r);

lon = atan2(recenter.y, recenter.x);

GPSCoordinate res(lat,lon, 0); //Assumes sea level

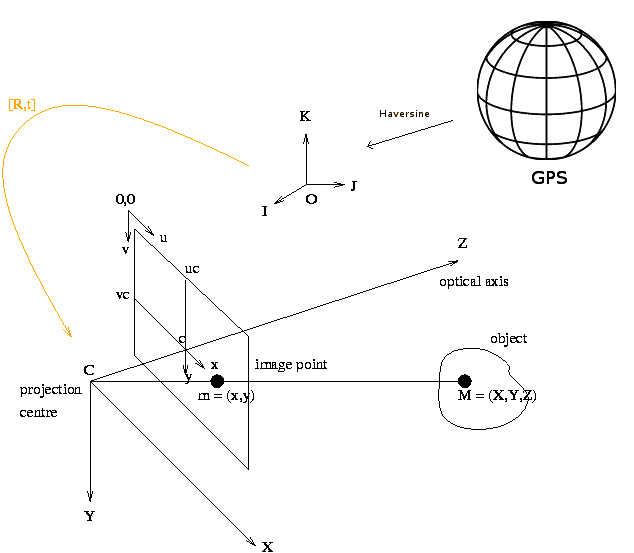
Return res;

}

## GPS to World Implementation Concerns

* The Haversine Formula assumes the earth is a perfect sphere with a radius of 6356.7 km. However, earth is more correctly approximated as ellipsoid, bulging along the center latitudes, with a variance of radius up to 20 km between the poles and equator. The Haversine formula produces a GPS to Cartesian conversion with a margin of error of .5% which may or may not be suitable for vision depending on what other approximations are made in the system.
* A more accurate conversion can be made using Versetis’ formula for distances on an ellipsoid, but this approximation is computationally expensive and requires the solving of simultaneous differential equations which is outside the scope of this paper. It is also too slow to accomplish in real-time on compact hardware.
* The Haversine Formula only accepts arguments in radians. Angles in degrees, minutes, seconds, have to be converted to a radian decimal representation before being input into the function.
* Different 3D graphics API may define the direction of coordinate axis differently. Check coordinate system orientations and switch arguments and signs as necessary.
* Floating point error may occur for very small angles such as cos(0.9999999) = cos(1) which is false. Use more accurate data types when working on small scales.
* Elevation is not assumed to vary in this representation. Aerial vehicles need a more accurate model.

# Section III: The World Centered to Camera Centered Transform

For the next step, which is converting from the World Coordinate System to the Camera Centered Coordinate system, world coordinates must be translated and rotated to align with the Camera Centered Coordinate system. For this transform, rotations and translations of homogenous coordinate matrices are used.

## 

**Figure 12.** World Centered to Camera Centered Transform Highlighted

## Rotation Matrices

For simplicity, an example of a rotation in two dimensions is given. The standard rotation matrix has the following form:



(10)

When the standard rotation matrix is multiplied by a 2-dimensional vector in the xy plane, the operation results in a new column vector which is a rotation of the original vector in the clockwise direction by an angle *θ.*

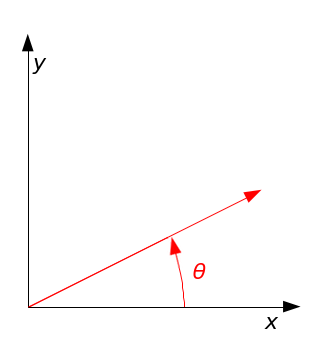
*For example:*

In the case of the unit vector on the x axis of length 1 in the form of a column vector:

a multiplication in the form of:



(11)

gives a new coordinate (x’,y’) which is the unit vector rotated by angle *θ.*

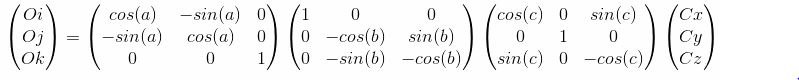


The result (x’,y’) =

**Figure 13.** Example rotation

For the rotation of the camera, simply generalize the same process into the three dimensions of the xy, the yz, and the xz planes, carrying out one rotation on each plane.

Equation (12) represents the total transformation as a composite of rotation matrices, each rotating in one plane. Column vectors (Oi,Oj,Ok) and (Cx,Cy,Cz) are points in the World Centered and Camera Centered coordinate systems. Angles a,b,c correspond to the roll, pitch, and yaw of the camera respectively.



(12)

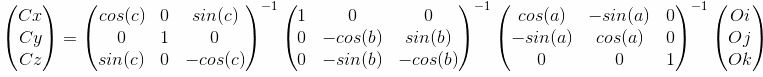
The signs of each trigonomic function depend on the axis directions that were earlier defined. Since World Coordinates are defined left handed, and Camera Coordinates are defined right handed, the signs of the Y and Z components of each rotation matrix are flipped as they are the ones that differ.

Since each rotation matrix is a square (or regular) matrix, each individual rotation can be combined by simply multiplying them. Carry out each rotation taking care of the order since A\*B\*C != C\*B\*C for matrices, meaning they do not commute.

Note that for ease of calculation, start from Camera Centered Coordinates and work backwards to World Centered Coordinates. Invert the resulting rotation matrix to get the transformation from World to Camera.

## Inverse Matrices

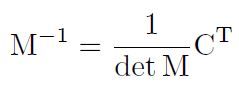
To find the goal which is a World Centered Coordinate in the Camera Centered Coordinate system, find the inverse of the former rotation equation (10). Finding the inverse of the rotation matrix once all three submatrices are multiplied is too difficult, so instead find the inverse of each plane rotation first before multiplying them in the reverse order. This process is illustrated in equation (13):



(13)

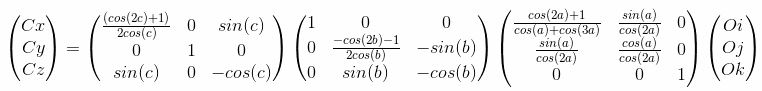
where (M)^(-1) is the inverse matrix.

The inverse of a matrix is given by the equation:



where detM is the determinant of matrix M, C is the matrix of minors of matrix M, and CT is the transpose of the cofactor matrix of M.

Applying the matrix inverse equation to each rotation matrix in equation (13) through a computer produces a new matrix:



(14)

arriving at an expression that takes a World Coordinate as input and outputs the same point rotated on to the Camera Coordinate system.

## Homogenous Coordinates

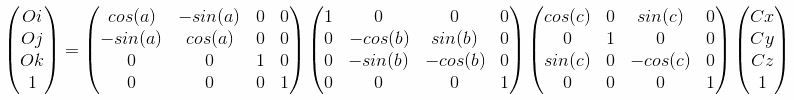
Now that an expression for rotations to and from the Camera and World has been derived, a problem surfaces. If a programmer were to implement our rotation transform in software as currently formulated, there would be severe memory and CPU bottlenecks since every coordinate is stored as point cloud data, and the program would have to calculate the rotation of every single point in the coordinate system. Additionally, the transformation matrix and its inverse are too large to be computed every time. A real-time system such as what is needed on a self-driving vehicle, needs something better.

Homogenous coordinates (or projective coordinates) are a coordinate system in which points on the same ray are considered to be the same point as long as they are a scalar multiple of each other. That is to say, the point (1,1,1) is equal to (2,2,2), and (3,3,3) in homogenous coordinates since each one is a simple scaling of the others, and all lie on the line through (1,1,1) and the origin. All three of these points can be represented as (1,1,1,w) where w is the homogenous scaling factor.

Homogenous coordinates of 3 dimensions are not easily visualized as they operate on 4-dimensional space, leaving three dimensional “shadows”. But the matrix operations are the same. Essentially, converting the transform into homogenous coordinates would allow the program to calculate the rotation of the coordinate system as rotations of a collection of rays instead of a cloud of points.

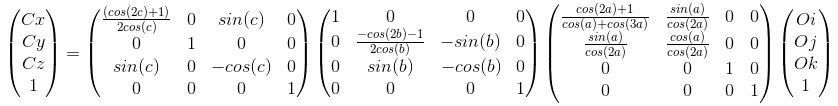
### Conversion to Homogenous Coordinates

The conversion of a 3D coordinate to homogenous coordinates is done by adding a 4th row to each coordinate triple. This row contains the *w* component which is the displacement in projective space. Equations (12) and (14) converted to homogenous coordinates are:



(15)

*Equation (10) in Homogenous Coordinates*



(16)

*Equation (11) in Homogenous Coordinates*

### Advantages of Homogenous Coordinates

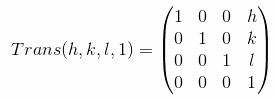
* CPU/Memory efficiency is improved by 30% or more.
* All spatial transformation matrices become regular and can be composited through multiplication.
* Composited transformations need be precomputed only once outside the program, with only the result used during program execution.
* Coordinate transform matrices can be represented with the same data type as coordinate triples in homogenous coordinates.
* Points at infinity can be represented as a coordinate with a *w*= 0 homogenous scaling factor, allowing certain perspective transforms to be simplified.

### Disadvantages of Homogenous Coordinates

* 4-dimensional spatial reasoning is harder to visualize.
* Pythagorean theorem homologue and distance formula must be modified.

## Translation Matrices

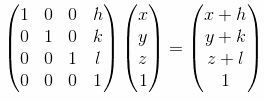
Although the world coordinate system has successfully been rotated to lie on the camera coordinate system and the rotation matrices converted to homogenous coordinates, the transform is not yet complete. The rotation transform is only correct assuming the camera had no displacement from the origin of the world coordinate system, meaning the origins of the camera and world coordinate systems coincide. Equations (15) and (16) do not hold true if the vehicle has traveled any distance from the origin. To fix this, translate the coordinate system by the displacement of the vehicle in world coordinates before rotating.

In homogeneous coordinates, the three-dimensional camera translation is given by the translation matrix:

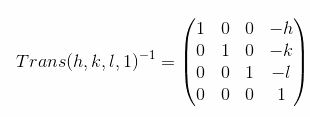
(17)

where *h* is the translation of the camera in x, *k* is the translation of the camera in y, and l is the translation of the camera in z.

If the translation matrix is multiplied by a coordinate triple in homogenous coordinates, it returns the same coordinate translated by the values of h,k,l.

(18)

The inverse of the translation matrix, found in the same way as the inverse rotation matrices, is:

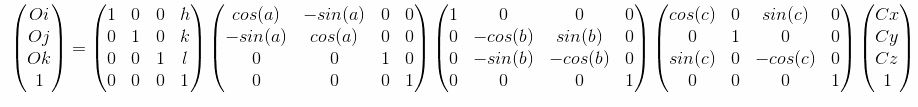


(19)

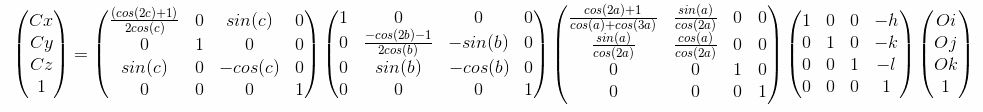
Which is simple to confirm through inspection.

## Appending Translation Matrices

Because the rotation matrices were previously converted to homogenous coordinates, all transformation matrices for both rotation and translation are square matrices. This fact allows us the creation of a composite of both types of transformations by concatenating the translation matrix (17) and the inverse translation matrix (19) directly on to equation (15) and (16) through a simple multiplication. Note that in equation (15) the matrix is concatenated on to the beginning, and for equation (16) the matrix is concatenate on to the end. This is because matrix multiplication is not communitive, so the order of the transformations must be reversed for the inverse operations.

(20)

*Eq.(15) with translation matrix appended*



(21)

*Eq(16) with inverse translation matrix appended*

## Camera Matrix

The Camera Matrix is a transformation matrix of the form:



Where R is the rotation component and T is the translation component.

Equation (20) is of this form once multiplied and reduced, meaning that it forms a single compact matrix that contain all information needed to translate from the Camera to World Coordinate systems.

Multiplying all the transformation matrices in equation (20) together results in a composite matrix of this form which is the particular Camera Matrix as defined by the axis directions of the World and Camera coordinate systems.

*Eq(22)*

*The expanded form of the Camera Matrix is written on the next page.*

The Camera Matrix is useful because it only needs to be calculated once. Once simplified, the result only needs to be implemented once in software without the need of repeating all the previous calculations every time a coordinate is transformed between systems.

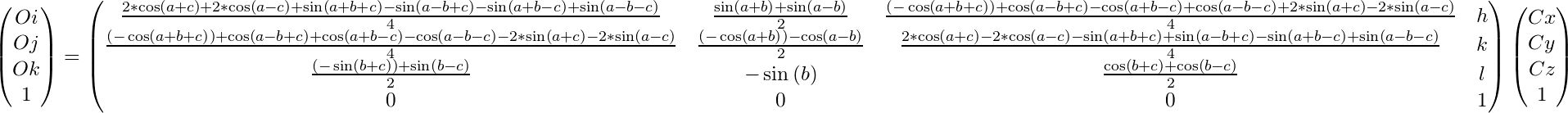
## Reversal

The reverse transform which gives a transformation from the World to the Camera coordinate system is simply the inverse of the Camera Matrix which can more easily be found as the product of all matrices in equation (21) instead of deriving it by taking the inverse of equation (20).

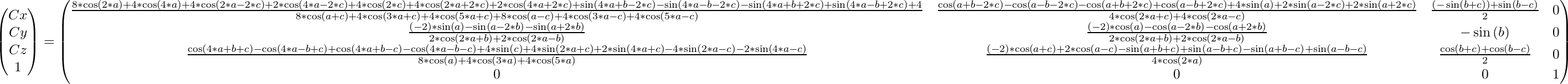
*Eq(23)*

*The expanded form of the Inverse Camera Matrix is written on the next page. Note that the translation term has not been multiplied as the resulting matrix is too large to fit on the paper.*

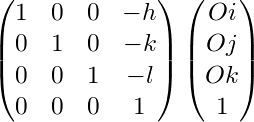
## Expanded Camera Matrix (Camera Centered to World Centered Transform)

(22)

## Expanded Inverse Camera Matrix (World Centered to Camera Centered Transform)

(23)

`



## World Centered to Camera Centered Pseudo code

CameraTriple WorldtoCamera (WorldTriple input, float roll, float pitch, float yaw, WorldTriple cameraLocation)

{

CameraMatrix CM(roll,pitch,yaw, cameralocation); //Initialize camera matrix depending on the camera position and orientation. Implementation is defined in EQ (19)

CameraTriple res();

CameraTriple res= CM \* input; //Multiply input point by camera matrix to carry out transform

Return res;

}

## Camera Centered to World Centered Pseudo code

WorldTriple CameratoWorld (CameraTriple input, float roll, float pitch, float yaw, CameraTriple cameraLocation)

{

InverseCameraMatrix CM(roll,pitch,yaw, cameralocation); //Initialize inverse camera matrix depending on the camera position and orientation. Implementation is defined in EQ (18)

WorldTriple res();

WorldTriple res= ICM \* input; //Multiply input point by inverse camera matrix to carry out transform

Return res;

}

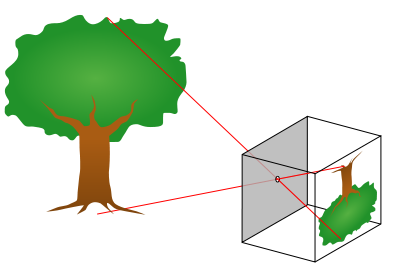
## Implementation concerns

* The fully expanded forms of the Camera and Inverse Camera Matrix are very large. Multiply the component matrices in software and save the result as a constant on program initialization.
* Floating point error may occur for very small angles such as cos(0.9999999) = cos(1) which is false. Use more accurate data types when working on small scales.
* This formulation of the Camera Matrix is unique to the definition of coordinate system directions. Equation (9) and all subsequent rotations have different signs depending on the axis formulation. Each camera matrix is unique to coordinate system formulation.

# Section IV: Camera to Camera Plane Projection

## Pin hole Camera Model

Any computer vision system operates on the pinhole camera model. The pinhole camera model is a mathematical model that describes the relation between points in 3D space and their projection on a 2D plane after passing through a small aperture. Incoming light from a scene passes through the aperture and is projected a certain distance on to a 2D camera plane at a distance given by the camera lens focal length.



**Figure 14.** The Pinhole Camera Model

## Projection of point on to a plane (perfect camera)

For a theoretical camera with no optical distortion, the location of a 3D Camera Coordinate is found on the camera plane by locating the point where the ray stretching from the camera origin to the Camera Point intersects with the Camera Pixel plane. This point gives the coordinate pair (mx,my) which is the location of a camera coordinate on the actual camera image in pixels. The Camera plane coordinate system is designed as follows:

A picture containing object

Description generated with high confidence

**Figure 15.** The Pinhole Camera Model Coordinate System

A close up of a black background

Description generated with high confidenceLooking at the coordinate system from the top down, the ray forms two similar triangles on both sides of the camera plane. The hypotenuse of the triangles is formed by the projection line (marked in green). To find the point on the y axis of the camera plane where the projection line intersects, one only needs to solve for the lengths of the similar triangles knowing that distance ***f*** is the focal length of the camera.

**Figure 16.** The Pinhole Camera Model viewed from above.

Let the pixel size of the camera plane be wP x hP meters and the image size be WIDTH x HEIGHT in meters. WIDTH x HEIGHT referring to the physical size of the camera image and wP and wH to the physical size of each camera pixel. If an object in the Camera Centered Coordinate system is known to be L meters high and at a distance *d*, the image on the camera focal plane will be L \* *f*/d meters high, with the objects’ physical size and projection size giving us the side lengths of the similar triangles x1/x3 and *f*/y1. To solve for the location in y, repeat this process looking at the coordinate system form the left.

Putting it in a more compact form, if given the location of an object as a camera coordinate triple (Cx,Cy,Cz), the location of said object in the camera pixel plane is given by:

(24) ) , )

## Camera Plane Projection Pseudocode

CameraTriple (CameraTriple input, float focal length, float WIDTH, float HEIGHT, float d. float wP, float hP)

{

Float mx my;

mx= ((WIDTH/2) + input.Cy \* (f/(d\*wP))); //EQ24

my= ((HEIGHT/2) + input.Cz \* (f/(d\*hP))); //EQ24

CameraPixel res(mx,my);

Return res;

}

## Implementation Concerns

* Most parameters are physical quantities found analytically. Camera parameters should be set as a configuration using a config file and are unique to the individual camera.
* Floating point error may occur for very small angles such as cos(0.9999999) = cos(1) which is false. Use more accurate data types when working on small scales.
* Camera pixel sizes must be measured by hand.

# Issues encountered and continuing research

## Camera Aberration & Distortion

The camera plane projection only works with a perfect theoretical camera, free of any distortions. In reality, all cameras suffer from innate tangential and radial distortions unique to each camera lens and current zoom level. Software techniques exist that can correct the optical distortion to allow for accurate camera plane projection, but they are outside the scope of this paper.

A picture containing clipart, black, object

Description generated with very high confidence

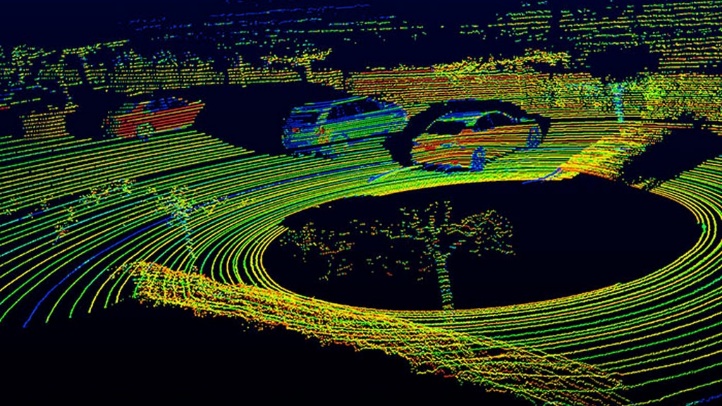
# 

**Figure 17.** Different types of camera distortion

In the future, implementations of the projection to Camera to Camera Plane projection should take camera distortion in to account.

## Reprojection and the distance problem

The reverse of the plane projection transform, which finds a 3D coordinate in Camera Centered Coordinates based on the location of the camera pixel plane depends on knowing the current distance between the camera and the imaged object. However, distance information cannot be found from the single camera’s information alone. A LIDAR system, laser range finder, or a parallax based binocular vision system are needed to determine distance.

A group of people in a store

Description generated with high confidence

**Figure 18.** A Binocular Vision System produced a depth map

**Figure 17.** Image of LIDAR telemetry from a self-driving car

## Software Validation

Because of the nature of the transform software, it is currently difficult to validate it without a full implementation in a fully function vision system or 3D simulation. Future work would attempt to compartmentalize each transform and validate for correctness and performance.

# Bibliography

“Depth Map from Two Images.” *OpenCV: Depth Map from Stereo Images*, docs.opencv.org/trunk/dd/d53/tutorial\_py\_depthmap.html.

“Projection of a Point on to a Plane.” *Projection of a Point onto a Plane, Projection of a Point onto a Line*, www.nabla.hr/CG-LinesPlanesIn3DB5.htm.

K.N. Srinivasa Rao, The Rotation and Lorentz Groups and their Representations for Physicists (Wiley Eastern Limited, New Delhi, 1988).

“Camera Calibration With OpenCV¶.” *Camera Calibration With OpenCV - OpenCV 2.4.13.6 Documentation*, docs.opencv.org/2.4/doc/tutorials/calib3d/camera\_calibration/camera\_calibration.html.

*Alexandrov, Pavel S. (1968), Lecture notes in analytical geometry, Russian, Science Publishers, Moscow*

*"CORRECTIVE OPTICS SPACE TELESCOPE AXIAL REPLACEMENT (COSTAR)". Retrieved 2015-07-18*. http://www.ball.com/aerospace/programs/hubble-space-telescope

Steger, Carsten; Markus Ulrich; Christian Wiedemann (2018). *Machine Vision Algorithms and Applications* (2nd ed.).

[*van Brummelen, Glen Robert*](https://en.wikipedia.org/wiki/Glen_Robert_van_Brummelen) (2013). [*Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry*](https://books.google.com/books?id=0BCCz8Sx5wkC&pg=PR7). [*Princeton University Press*](https://en.wikipedia.org/wiki/Princeton_University_Press). [*ISBN*](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [*9780691148922*](https://en.wikipedia.org/wiki/Special:BookSources/9780691148922). 0691148929*. Retrieved 2015-11-10*.

Richard Hartley and Andrew Zisserman (2003). Multiple View Geometry in computer vision. Cambridge University Press. [*ISBN*](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [*0-521-54051-8*](https://en.wikipedia.org/wiki/Special:BookSources/0-521-54051-8).

# Acknowledgments

*A special thanks to:*

*Doctor Laurie Anderson of UWB school of STEM for paper revision and layout suggestions.*

*Doctor Warren Buck of UWB school of STEM for help with calculations and mathematical revision.*

*Doctor Tyler Folsom of UWB school of STEM for guidance with project direction and guidance with computer vision software.*