Throughout, all frames are right handed.

First look at “\TriangleRayIntersection\TriangleRayIntersection.m” note that it takes xyz of a ray origin and a ray direction.

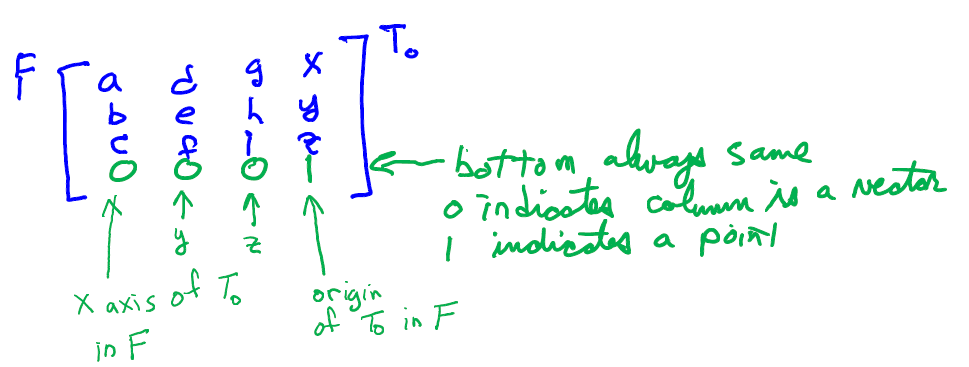
function [intersect, t, u, v, xcoor] = TriangleRayIntersection (...

orig, dir, vert0, vert1, vert2, varargin)

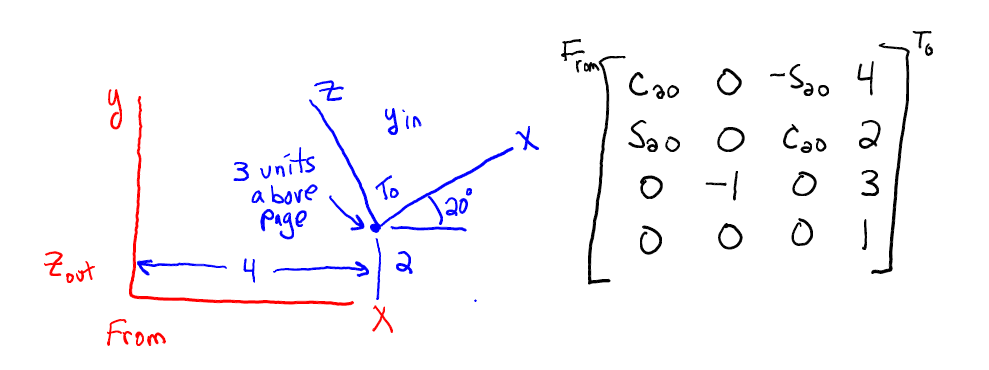
using the origin and direction of the ray, it finds the intersection point of the ray (laser).

Now look in “\TriangleRayIntersection\FindTheHit.m” note that it accepts a 4by4 matrix called laser (a Transformation matrix) and calls TriangleRayIntersetion.m it sends column 4 of laser as the orig and column 3 of laser as the dir.

Now a quick review of Transformation matrices.



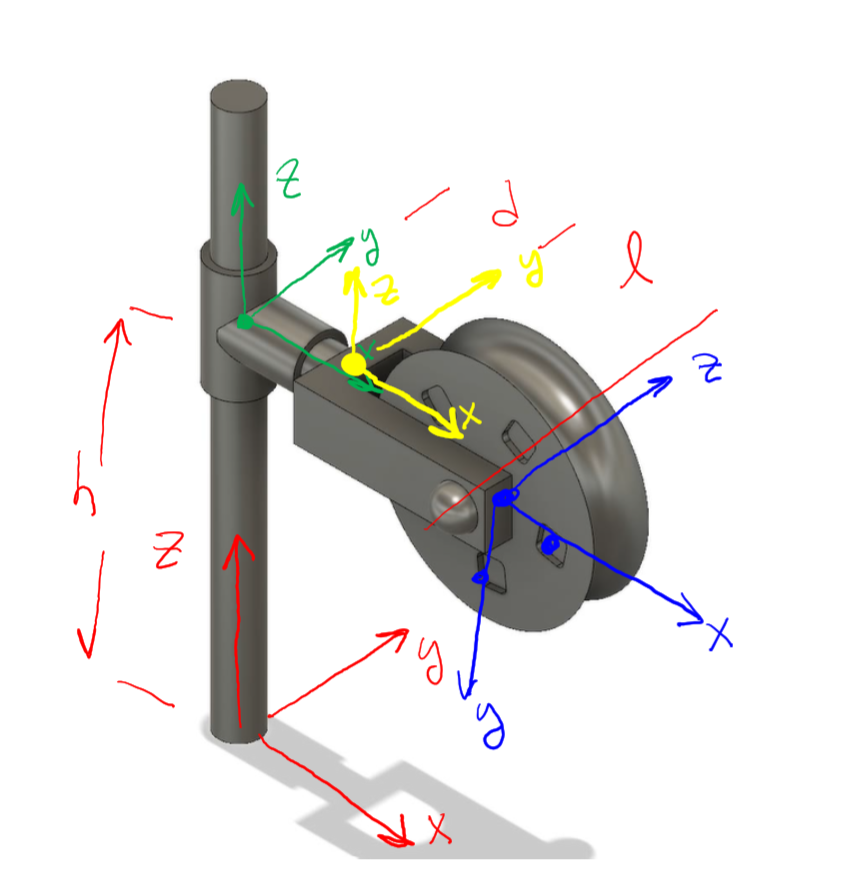
The fourth column of a Transformation matrix is the origin of the To frame measured in the From frame. The third column is the direction of the z axis of the To frame measured in the From frame. If one axis of From and To are parallel, it is easy to determine the values for the Transformation matrix.



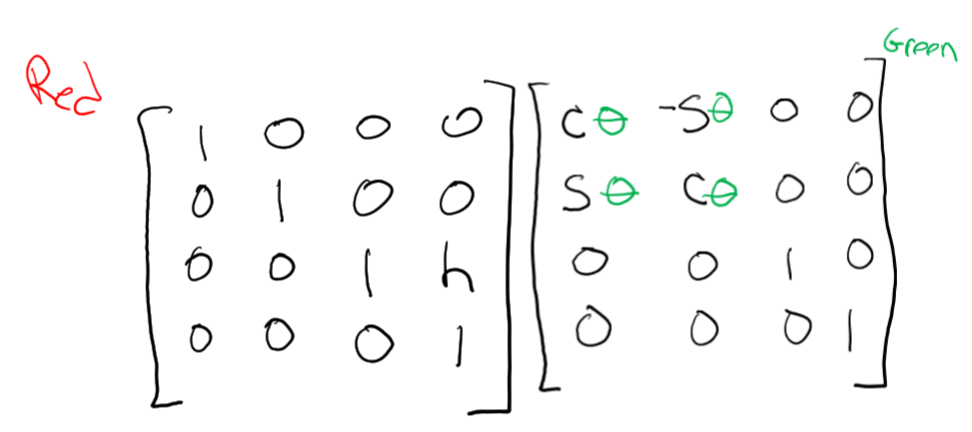
Here the Red is a frame called From, the Blue is a frame called To. We want to move from the From to the To. To go from red to blue you move 4 in red x 2 in red y and 3 in red z. Note this is the 4th column of the From-To Transformation. You must ALWAYS move to the To first. Next look at the blue X axis, it is pointed Cos(20) along red x, Sin(20) in red y and nothing in red z. This is the column 1 of the Transformation. The blue y is opposite the red z so the 2nd column is 0,0,-1. The blue z is -Sin(20) along red x, Cos(20) along red y and nothing along red z, hence the 3rd column as shown.

Normally we memorize the rotations about x y and z because they are used very often. BUT they can be “derived” using the same method as above.

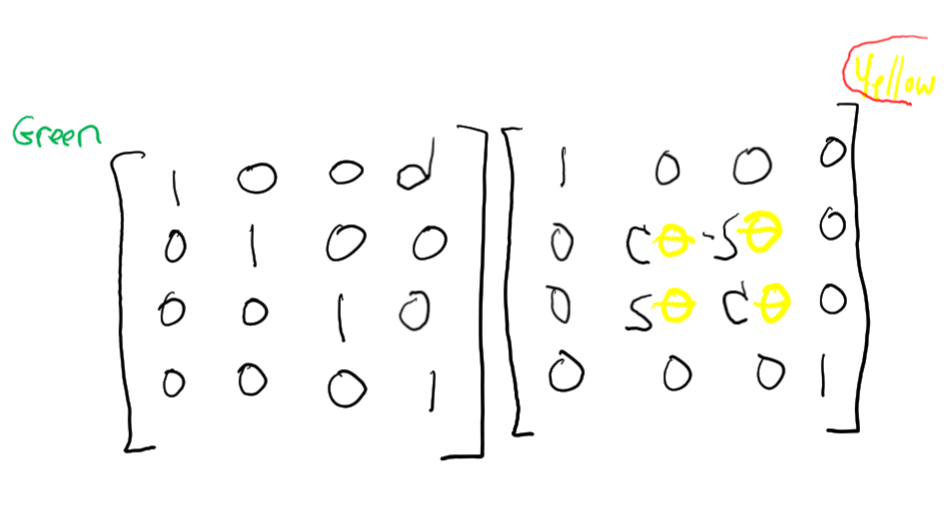
When the red and blue do NOT have one axis aligned you typically build them up one rotation at a time. First watch the video SpoolAndFork.avi. I emailed it to you. Put a frame on each moving part. Red is fixed to the ground. The x y z are kind of arbitrary but you should align them conveniently. The figure below shows how to do this if ALL the angles and values are zero (kind of, bear with me). Red is fixed on the ground, you would put red on the “map”. Green is in the center of the post, my drawing isn’t perfect. The yellow is in the fork in the center as well. The blue is way off but I intended it to be in the center of the spool aligned with the axle it spins on.



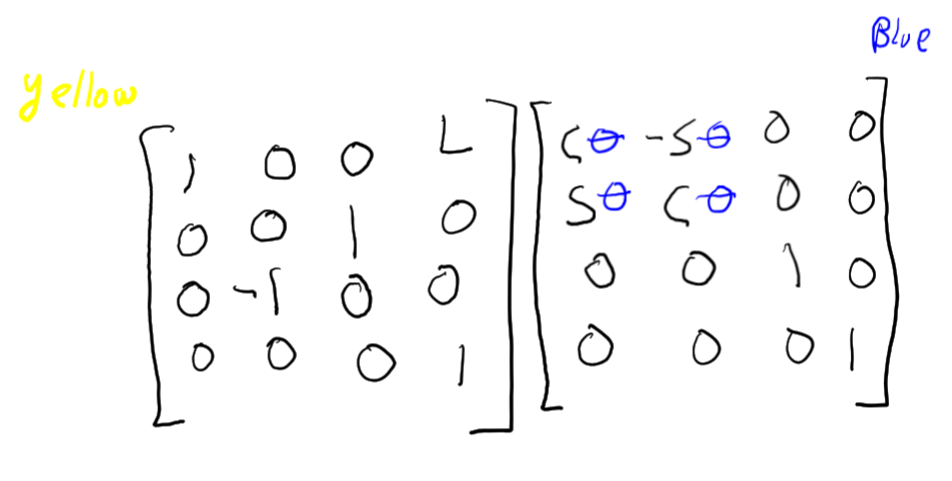
Now move from Red to Green, translate h along red z. Then rotate about red z. this means is positive if view from the top of the fixed post.



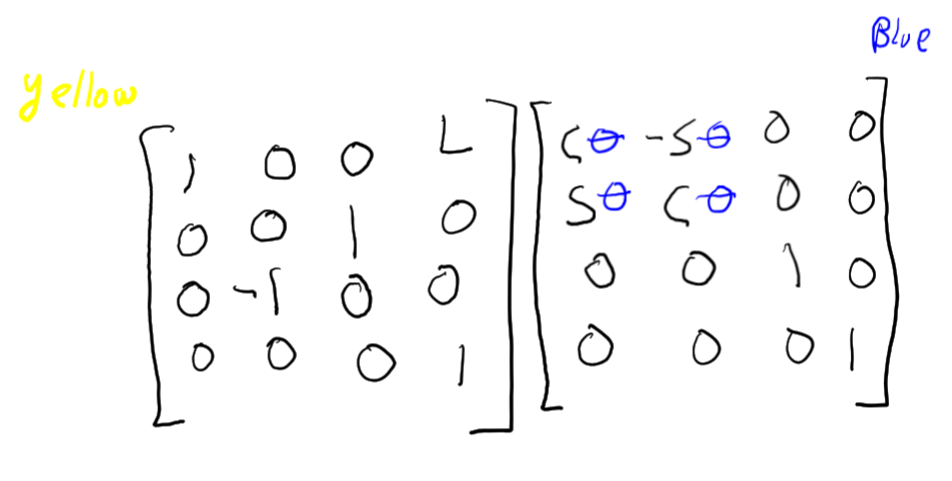
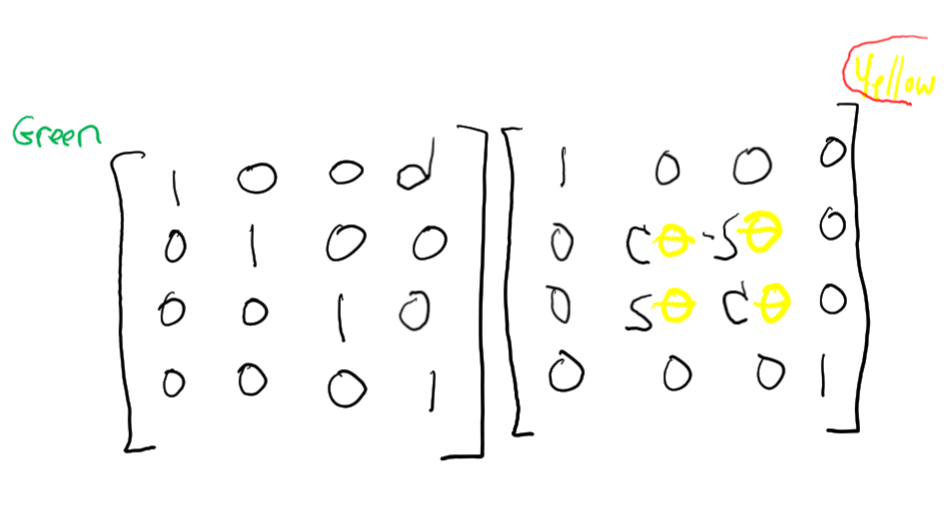
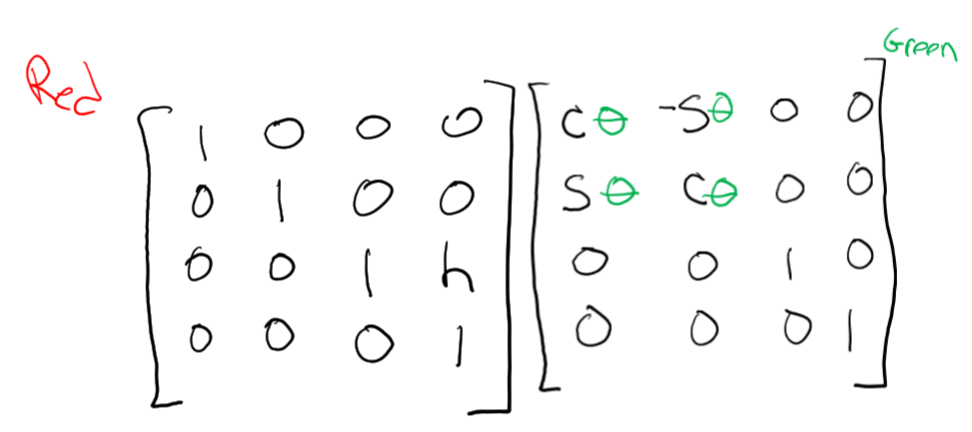
Note that although you have “moved” the yellow, and blue locations RELATIVE to the Green are exactly the same as they started out to be. So you can easily move FROM the Green to the Yellow by moving d out the Green x, then spinning a about the Green x.



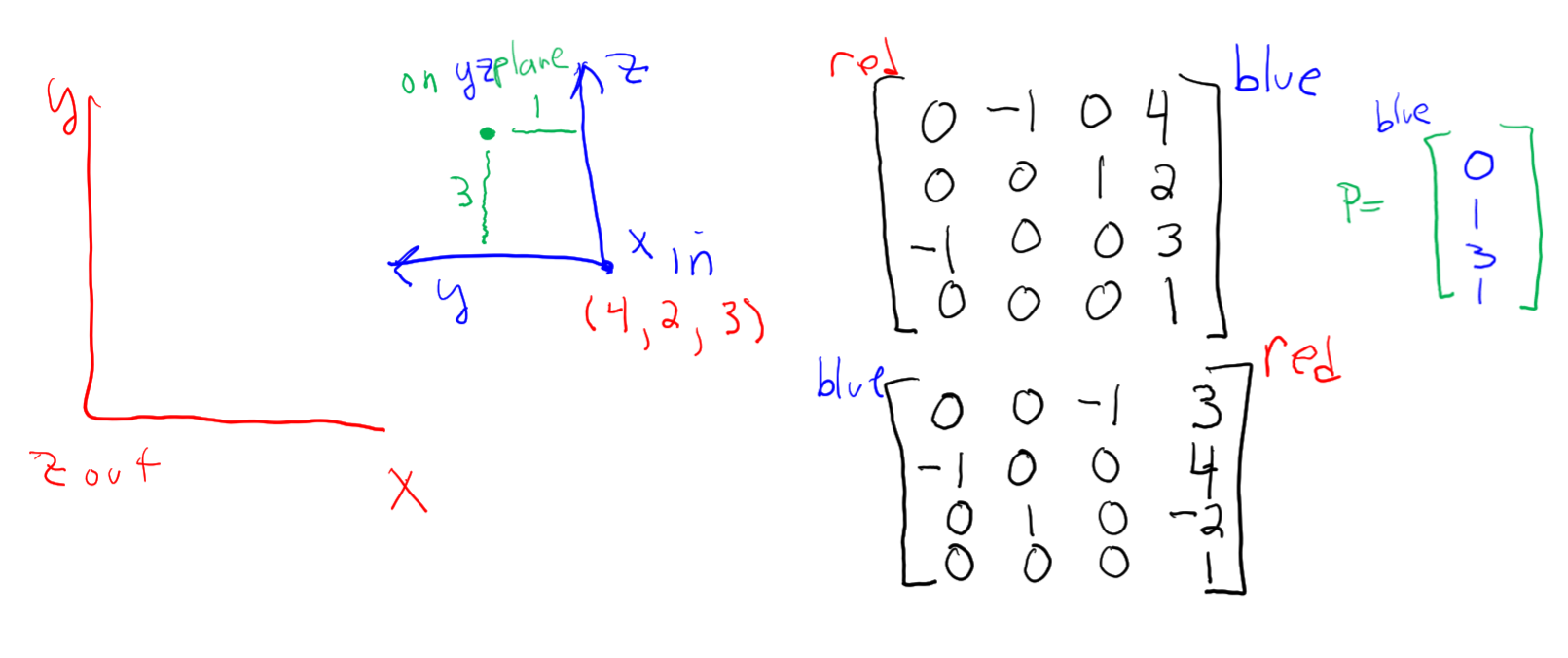
Like before the location of the blue relative to the yellow has not changed so you can move yellow to blue by moving L in yellow x and flipping the blue axes around so they line up as drawn. After lining up the blue you can put on a



Now just put them all together:



One thing that confuses people is that when you “create” the equations you are thinking about moving cosysystems from the reference (often fixed co sys) TO the moving or end co sys. HOWEVER once you have the equation it is used to convert points measured in the moving or end co sys back to the reference co sys. Here is an example. The figure shows a RED and BLUE co sys. It also shows a GREEN point. Note that the green point is located relative to the BLUE frame as 0,1,3. If measured in the RED it would have coordinates of 3,5,3. Also note that the two co sys matrices are determined from inspection first moving red to the blue then moving blue to the red. Note also if you multiply the two matrices together you get identity because they are inverses.



Now do this or do this so to find the co sys you move from the “beginning” cosys toward the end, but to convert points you “kinda” go backwards. Use the superscripts to give you a clue. You will find that different people use different scripts so that can be confusing too. For example some use scripts like this: and others will do this so you need to read their papers carefully to ensure you know what the upper script and lower scripts are.

Back to the lidar code. If you think of the map as having a coordinate system you can relate the location relative to the map cosys. You could make things relative to the Earth if you want, but all the transformations between the Earth cosys and the map are constant so you could do everything relative to the map and it is a simple operation to convert to the Earth. So let’s define the lidar relative to the map.

Let the map origin be anywhere you want, let’s say the lower left corner of the map. Laying the map out flat to the right is normally East (x axis), the top is North (y axis) up from the paper is the Vertical (z axis). To move to the lidar frame, line up the DRONE (similar to the video, the lidar is ON THE DRONE so you locate the Drone then located the lidar relative to the Drone) with the map origin. Now the x y z of the Drone is arbitrary but most papers treat the x as the direction a camera mounted on the Drone points (the camera looks forward). The Z of the Drone is usually through the belly pointing toward the ground. The y axis is therefore out the right wing (if the Drone had wings, the notation is borrowed from classical aircraft). When initially aligned using the “common” axes, the Drone is pointing its camera East and it’s belly is up. Move the drone (remember you always move before rotating) 3 distances X, Y, Z relative to the map. These are the LOCATIONS of the Drone center relative to the map. Location is the same if its belly up or belly down. Next flip the Drone over so its Belly is down as desired by rotating 180 about x (you could also rotate about y there are multiple ways to get there but what follows assumes you rotated about x). Now the Drone is belly down facing East. Next most put on a yaw rotation about Z (which is pointed at the ground now) causing the nose to turn right or South. Then pitch around Y (right wing) causing nose to go up. Finally roll about X (which is south and up, assuming small positive yaw and pitch) causing the right wing (if it had one) to go down.

Now look at “main.m” on line 12 it defines the drone location relative to the map of [x y z yaw pitch roll] of [10.1 10.1 40 0 0 0] look in “TRDrone.m” and you can see where the cosys matrix is defined. You will also note that I forgot to flip the drone over because I forgot to rotate 180 about x. Oops!

Now look at “LidarSweep.m” note that it receives the location of the Drone. Also note that the z of the drone is vertical because I forgot to flip him over. The function computes angles for the laser beam beginning at 0 and going to 191 degrees (not sure why I chose 191 degrees). Now if the yaw pitch and roll are not zero it is difficult to visualize what is happening so for now, assume the drone yaw pitch roll are zero, this means the drone x y z axis align with the map. For each angle for the laser beam I tack on a rotation about Y to get to the LASER co sys this means I am “moving” the Drone co sys into the Laser co sys by “spinning” about the Drone y which means the LASER x and z axis are no longer aligned with the map. So when I call FindTheHit the Laser Z axis is not perpendicular to the map, it is sweeping. Now what I cannot explain is if the lidar angle is 0 then the laser is pointed up and away from the map so I’m would expect the routine to not find a distance to the map. Hmm! Probably should run it a few more times but before I do, I should flip it so it is belly down so the laser will point toward the map. I think also I should begin the laser sweep at -90 degrees and go to +90. Let’s talk.