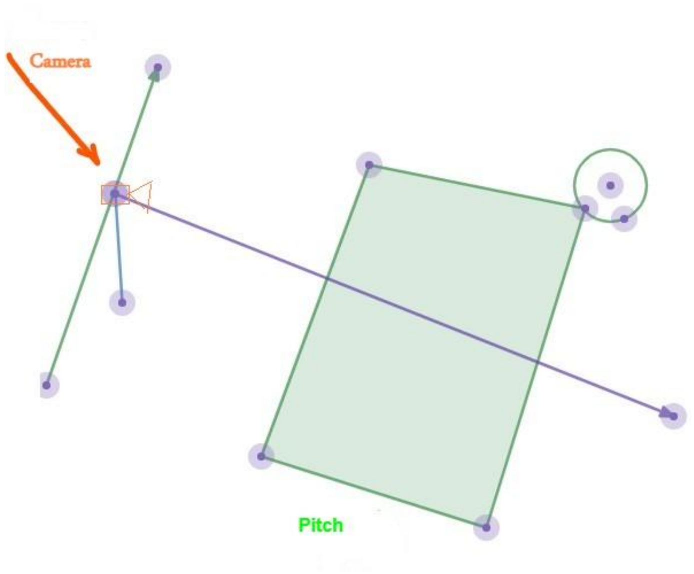
Geometric representation of the remapping soccer field

As describing some geometric operations is difficult using only comments in code, this slide can help to clarify the 3D reconstruction i used.

Camera and pitch position in real world



Map to camera space

To map from 2D image into the 3D camera space we use a basic geometric relations: x,y: points in image and X,Y: points in camera coordinate, f: focal length

Points on image: $x = f\frac{X}{Z},\ y = f\frac{Y}{Z}$

If we have camera center and f then the following matrix relations hold:

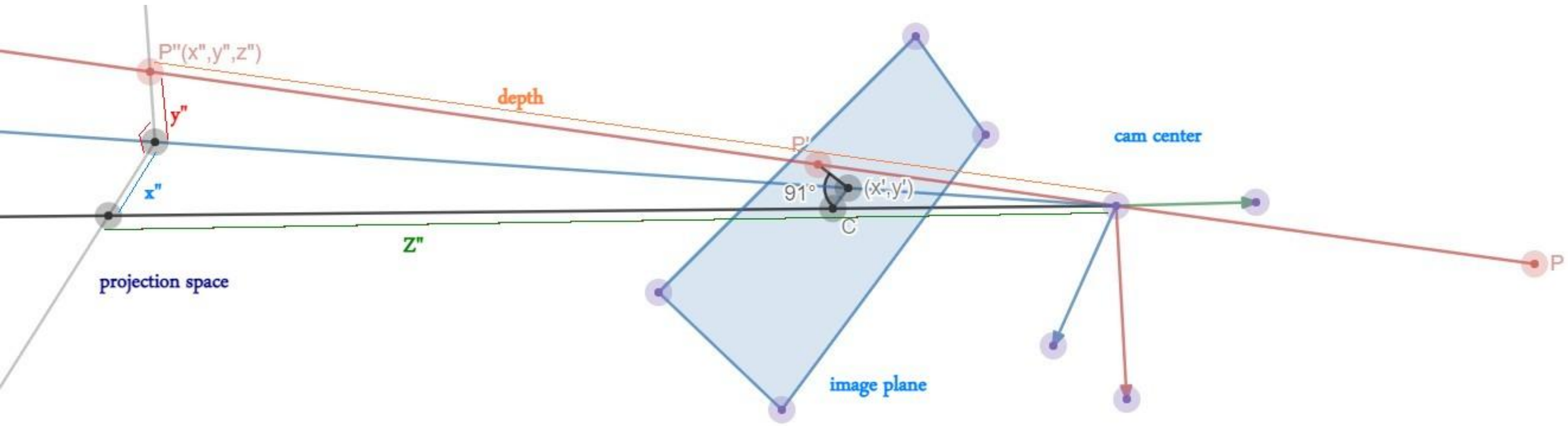
$$\boldsymbol{x} = K\left[R|T\right]\boldsymbol{X},\ \boldsymbol{x} = \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{w} \end{bmatrix},\ \boldsymbol{X} = \begin{bmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \end{bmatrix} \qquad K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Using the relation $x = PX,\ P = K\left[R|T\right]$

we can move from image coordinate to camera coordinate and vice-versa (P has inverse)

In this task, 2d image points and camera parameters is known, using this relation, we can calculate camera space.

Depth versus **Z** coordinate



Knowing Z axis is essential in transforming camera to the target coordinate

Calculate z

Projecting points to camera space, we have (X, Y) and **depth**.

To proceed further, I need to calculate Z from depth using the relationship between them: knowing Z is negative (images constructed on the back of the pinhole)

$$d = \sqrt{x^2 + y^2 + z^2}$$

$$z = -\sqrt{d^2 - x^2 - y^2}$$

Visualizing the reconstruction problem

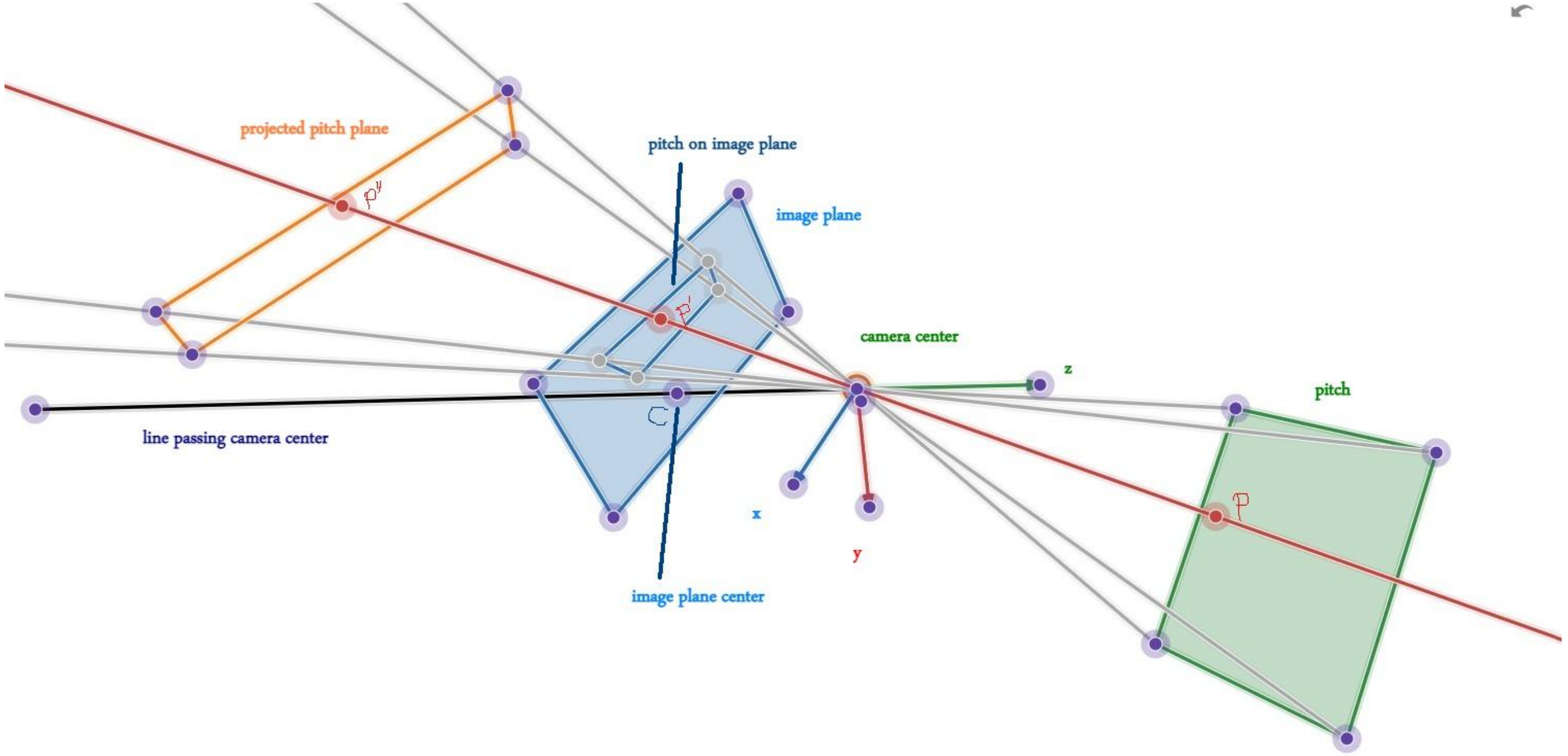
The core part of the problem is to get the players position at every frame and reconstruct the global camera coordinate first, Then transform it to the target coordinate.

Given the camera intrinsic parameters K and R, T, I assume that R,T are considered the camera movements (no need to recalibration); that means if a player do not move for a while (for some time), using the projection matrix **P = K[R|T]** I will get the same **(x,y)** and depth for that player if if camera moves.

Otherwise the problem will be different and we will need to use references (eg. lines in the pitche) and recalibrate the camera after any movement.

To do this, we should detect the lines, knowing their real size, match it with the target pitch lines that we created with the same scale as the real pitch. Then minimize the reprojection error subject to the camera relationships (grid search on focal length and improve other parameters iteratively).

Visualizing the reconstruction problem



Reconstructing pitch plane

I order to move from camera coordinate to target coordinate, we need to have projected pitch plane, in theory we can calculate projected pitch plane using the positions of 3 plates; then

$$v_0 = p''_1 - p''_0$$

$$v_1 = p''_2 - p''_0$$

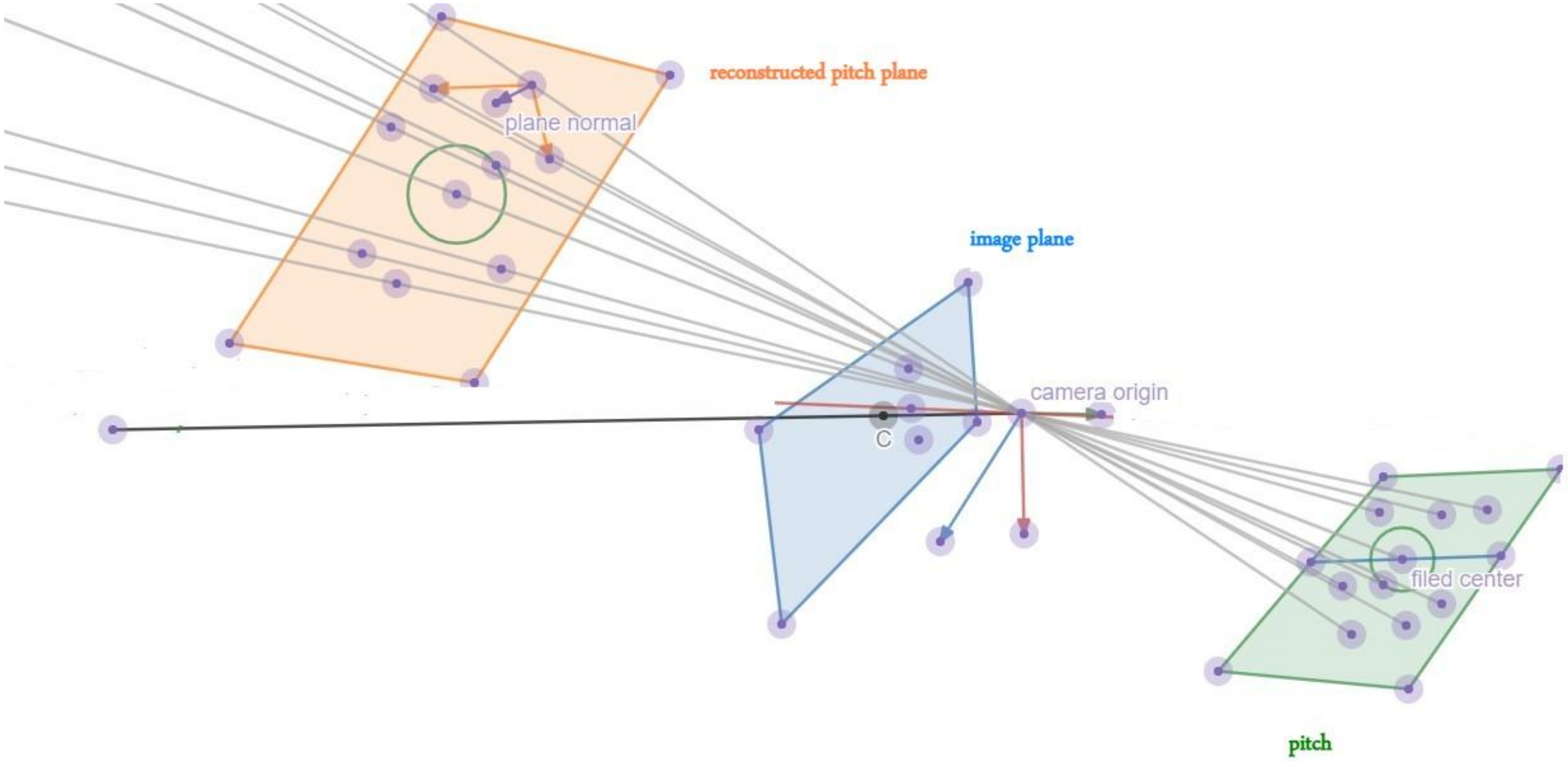
$$\vec{n} = v_0 \times v_1$$

$$plane\ equation: \left(p - p''_0\right).\vec{n} = 0$$

P” is projected player positions.
Note: Direction is important in cross product (not commutative)

A separate code is provided to explain an algorithm to restope pitch plane from noisy data

Reconstructing pitch plane



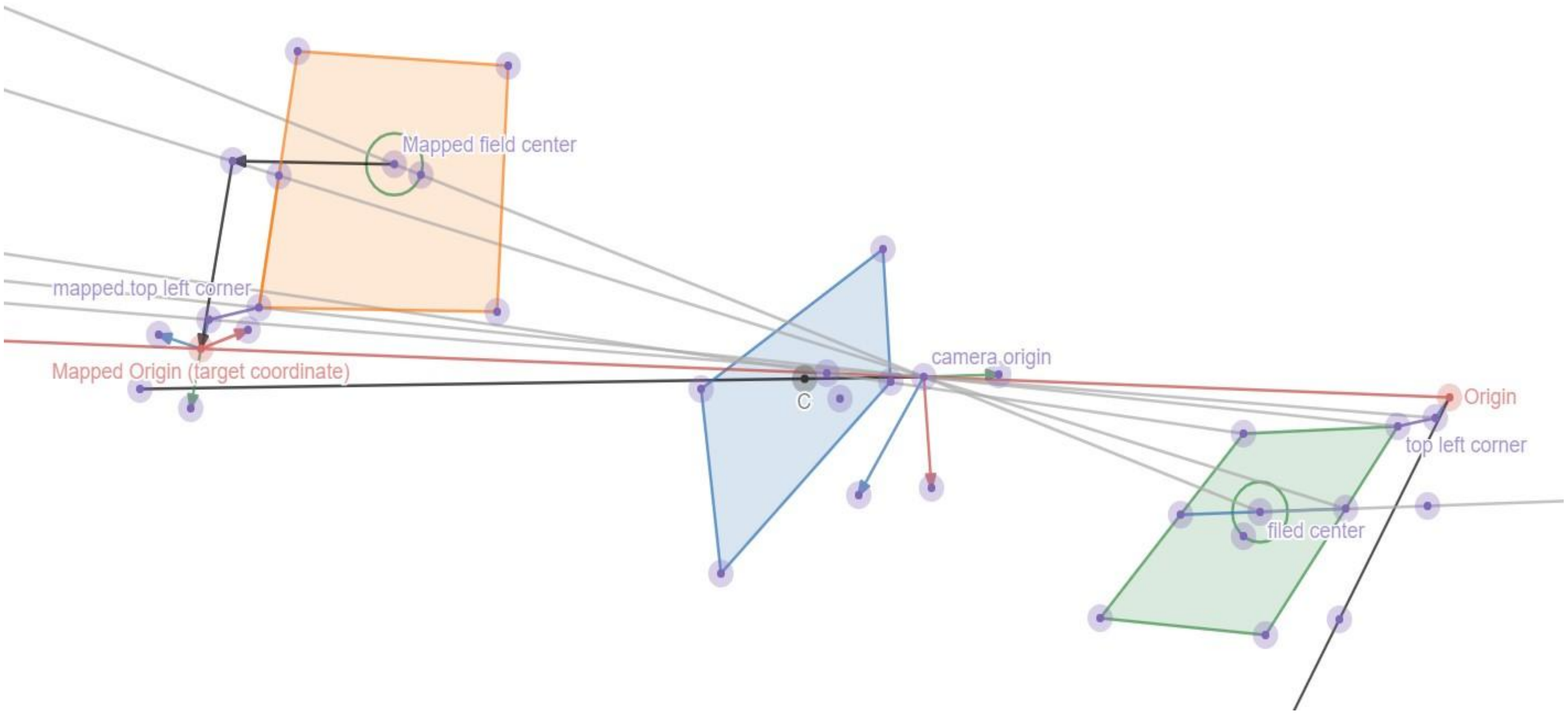
Target coordinate

Target coordinate will be with 50 units margins next to the top-left corner. That means the camera view should rendered accordingly.

y-axis of the target coordinate will be parallel to the normal of pitch plane.
z- axis will aligned with longitude of the projected pitch
x-axis will aligned with latitude of the projected pitch

First of all we need to find the position of the target coordinate in camera coordinate system;

Target coordinate



Find the position of the target origin

To solve this problem we can use the first frame (beginning of the match) where ball is placed exactly in the middle of the circle and two players are standing on the middle line very close to the ball.

Though this may not happen all the time, and players may not stand on the middle line, players and ball position is the only tools at hand, this could better handle by line detection.

Having the ball position exactly in the middle, two points parallel to the width side line of the pitch (parallel to the target x- axis), it is possible to calculate the target coordinate position.

Find the position of the target origin

Let **b** to be ball position at the first frame and **p** denote the players position. There would be a line passing **p** and **b** parallel to the middle pitch line.

$$l_1 = b_0 + t * \vec{d}_1$$

We need to know the direction **d1**. The direction can calculated using the players positions:

$$\vec{d}_1 = \left(p_2 - p_1\right)$$

The scalar parameter **“t”** will be half of the pitch latitude + 50 units margin.

Next I should calculate the line perpendicular to line **l1** and move parallel to the pitch longitude side line to the left:

$$\vec{d}_2 \cdot \overrightarrow{d_1} = 0$$

$$\vec{d}_2 \cdot \vec{n} = 0$$

This equations well led to two solutions (two opposite directions). To select the correct direction, it is enough to move to left side of pitch (right in the projected plane).

Find the position of the target origin

Having the equations at hand, we just need to start from **l1** and

Move in the direction **d2** with parameter **t** equal to half of the

Longitude of the pitch + 50 units:

$$\vec{d}_1 = \left(p_2 - p_1\right)$$

$$l_1 = b_0 + t * \vec{d}_1$$

$$\vec{d}_2 \cdot \vec{d}_1 = 0$$

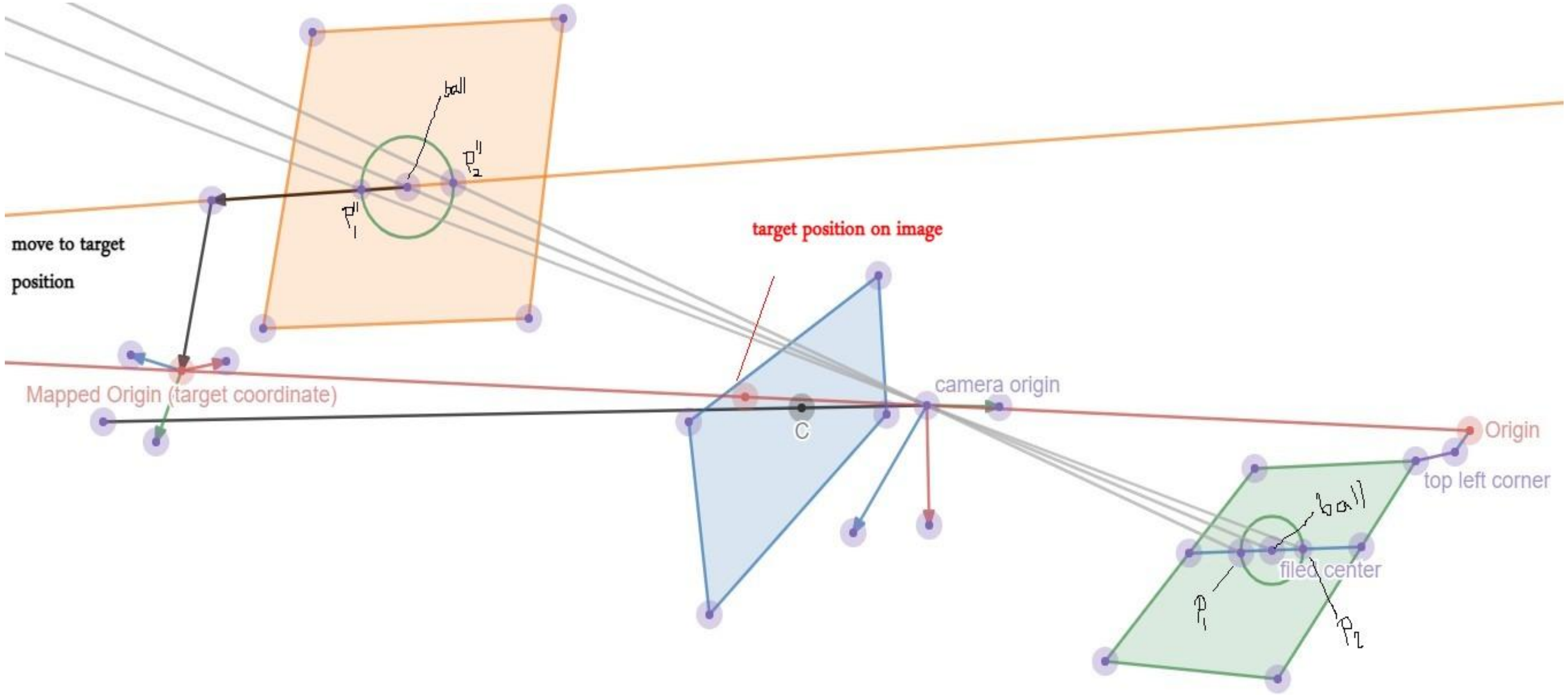
$$\vec{d}_2 \cdot \vec{n} = 0$$

$$O_t = l_1 + t * \vec{d}_2$$

Note: -d1 and -d2 are exactly parallel to the x and z axis of target coordinate, during calculations, d1, d2 will include errors. The solution can be improved by minimizing the translating back the player positions.

Note: directions are normal vectors (magnitude is 1).

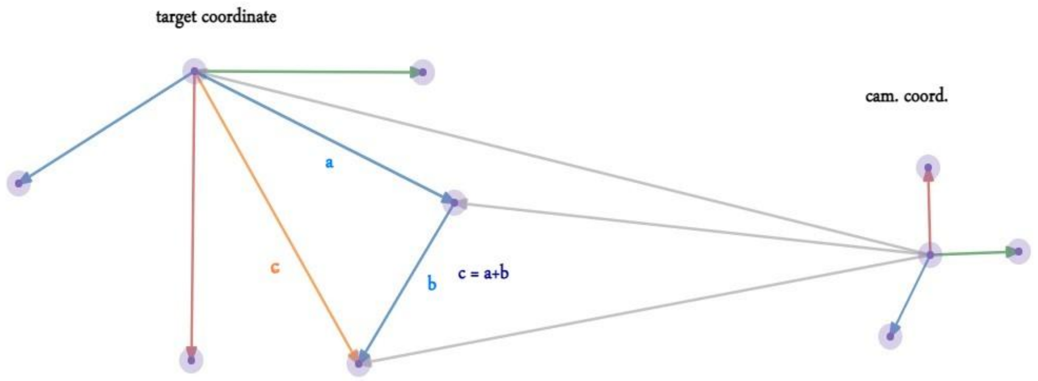
Get to target origin



Move to the target coordinate

Last part is to calculate **R’, T’** to change the coordinate.

Knowing the target origin coordinates in camera coordinates, this can simple done by coordinate translation, then having enough matched points, we can calculate R’,T’ and automate rest of the calculations.



Depth of player pose

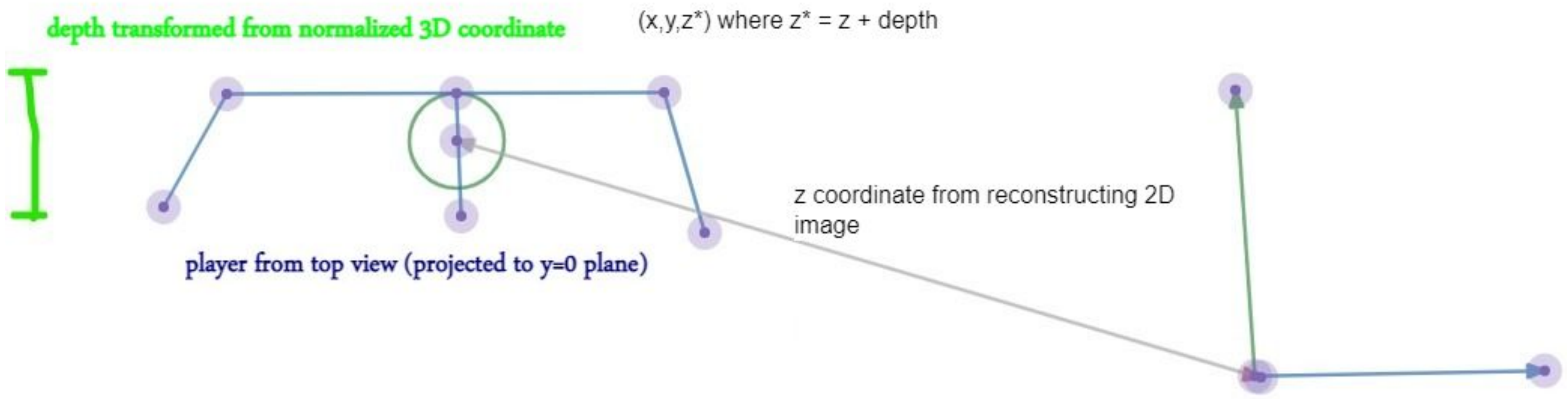
Until now we reconstructed player positions from given 2D coordinates into the target coordinates. Next problem to address is to calculate the depth of the player poses.

Depth of the poses cannot be solved without provided normalized 3d poses that could be output of a model or etc.

I handled the depth using PnP algorithm implemented in opencv by providing 3D joints (in normalized coordinate) and corresponding projected 2D joints (x- and y- in target coordinate).

Now if I use this rotation and translation matrix, I can transform standard coordinate to the target coordinate (x,y) and z will be the depth of player poses I am looking for.

Depth of player pose


depth transformed from normalized 3D coordinate
(x,y,z*) where z* = z + depth
z coordinate from reconstructing 2D image
player from top view (projected to y=0 plane)

Further improvements

Smoothing pose movement:

Optimizing the pose coordinates based on

- Differential (not let joints to move extra fast)
- Temporal move (make temporal change smooth and slow)

and

- keeping them close to the estimated points

I also provided a method to check the body bone sizes to remove wrong cases

Further improvements

During rotation and translation, we always have errors that leads to some sort of problems in making players straint

One solution to this problem is to use rays to map joint to pitch plane first and then

Create a perpendicular plane on that position, then again reproject player position to that plane.

This is like we pass a plane inside the player and reduce standing errors using that plane

Further improvements

3D reconstruction problem is prone to errors, one efficient way to reduce arrows is to use geometric tools and relations once in awhile and by that equations try to keep system stable and consistent

And there more to write about this.