Perspective-n-Point Problem :

Given a set of n 3D points in a world reference frame and their corresponding 2D image projections as well as the calibrated intrinsic camera parameters, determine the 6 DOF pose of the camera in the form of its rotation and translation with respect to the world. This follows the perspective projection model for cameras:

Resources :

ARUCO DETECTION:

Solve PnP problem:
Pose Estimation - https://www.youtube.com/watch?v=kq3c6QpcAGc
What the n in PnP means - https://www.youtube.com/watch?v=0JGC5hZYCVE
Problem Explained - https://www.youtube.com/watch?v=RR8WXL-kMzA
Detailed Explanation of equations - http://users.umiacs.umd.edu/~ramani/cmsc426/Lecture23_3Dpose.pdf
Grunett's method to solve P3P - https://www.youtube.com/watch?v=N1aCvzFll6Q&list=PLgnQpQtFTOGTPQhKBOGgjTgX-mzpsOGQX&index=17
RANSAC optimisation of equations https://www.ecva.net/papers/eccv_2020/papers_ECCV/papers/123470239.pdf
OpenCV parameter analysis: https://blog.fearcat.in/a?ID=00500-191fe72f-79a1-4377-8f2e-c0dea3f692b8
Otsu Thresholding: https://learnopencv.com/otsu-thresholding-with-opencv/
Quadrilateral Sum Conjucture : https://ria.ua.pt/bitstream/10773/23890/1/artigo.pdf

Optimisations used :

1. Rotational Matrix filter
2. RANSAC
3. Douglas-Peucker algorithm
4. Otsu thresholding
5. Subpix refining
6. Polygon/Quadrilateral Sum Conjuncture
7.

Plan of Action(Parameters to optimise) :

1) Adaptive Thresholding on the input image of the cv bridge

1. `detectMarkers()` :- `DetectorParameters object parameters`

a. **adaptiveThreshWinSizeMin(3) & adaptiveThreshWinSizeMax(23)** : parameters represent the interval where the thresholding window sizes (in pixels) are selected for the adaptive thresholding
b. **adaptiveThreshWinSizeStep(10):** `i`ndicates the increments of the window size
c. **adaptiveThreshConstant(7) :** parameter represents the constant value added in the thresholding operation

2) Contour filtering

a. **minMarkerPerimeterRate(0.03) & maxMarkerPerimeterRate(4.0) :** These parameters determine the minimum and maximum size of a marker, specifically the minimum and maximum marker perimeter.

EX: a image with size 640x480 and a minimum relative marker perimeter of 0.05 will lead to a minimum marker perimeter of 640x0.05 = 32 pixels

b. **Douglas-Peucker algorithm(** `polygonalApproxAccuracyRate(0.05))`: A polygonal approximation is applied to each candidate and only those that approximate to a square shape are accepted. This value determines the maximum error that the polygonal approximation can produce

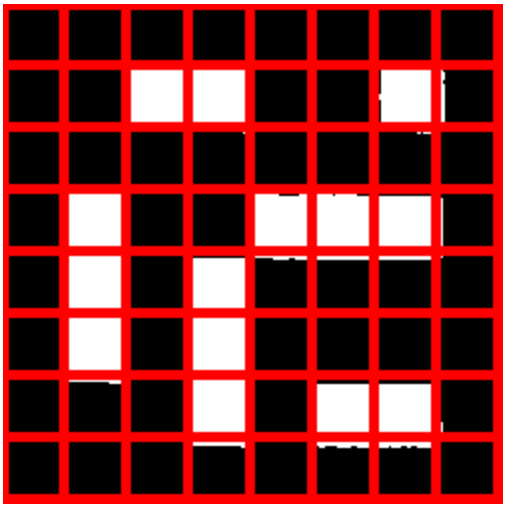
EX: f the candidate has a perimeter of 100 pixels and the value of `polygonalApproxAccuracyRate` is 0.04, the maximum error would be 100x0.04=5.4 pixels.

c. **minCornerDistanceRate(0.05) :** Minimum distance between any pair of corners in the same marker. It is expressed relative to the marker perimeter.

EX : Minimum distance in pixels is Perimeter * minCornerDistanceRate.

d. **minMarkerDistanceRate(0.05)** : Minimum distance between any pair of corners from two different markers. It is expressed relative to the minimum marker perimeter of the two markers. If two candidates are too close, the smaller one is ignored.
e. **minDistanceToBorder(3) :** Minimum distance to any of the marker corners to the image border (in pixels). Markers partially occluded by the image border can be correctly detected if the occlusion is small. However, if one of the corners is occluded, the returned corner is usually placed in the wrong position near the image border.

Note ::: If the position of marker corners is important, for instance if you want to do pose estimation, it is better to discard any markers whose corners are too close to the image border.



3) **Bits Extraction :**

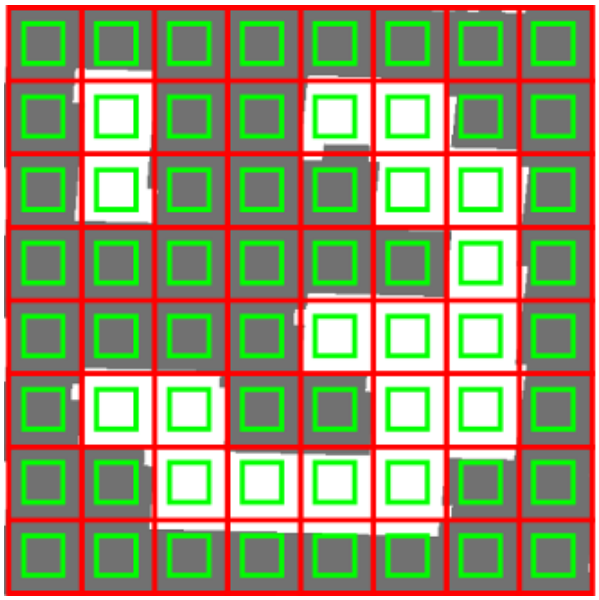
a. **markerBorderBits(1) :** This parameter indicates the width of the marker border. It is relative to the size of each bit. So, a value of 2 indicates the border has the width of two internal bits.

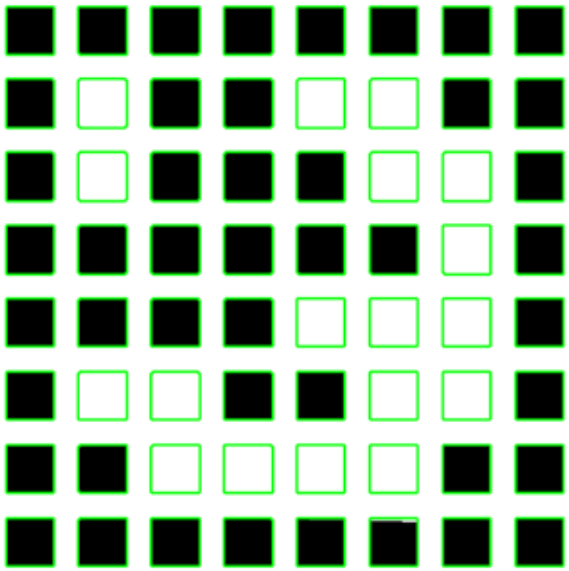
b. **Otsu thresholding(minOtsuStdDev(0.05)) :** determines the minimum standard deviation of the pixel values to perform Otsu thresholding.

c. **perspectiveRemovePixelPerCell(4) :** This parameter determines the number of pixels (per cell) in the obtained image after removing perspective distortion

d. **perspectiveRemoveIgnoredMarginPerCell(0.13) :** This parameter is relative to the total size of the cell.

EX : For instance if the cell size is 40 pixels and the value of this parameter is 0.1, a margin of 40*0.1=4 pixels is ignored in the cells. This means that the total number of pixels that would be analyzed in each cell would actually be 32x32, instead of 40x40.





4) Marker identification

a. **maxErroneousBitsInBorderRate (0.35):** The bits of the marker border should be black. This parameter specifies the allowed number of erroneous bits in the border, i.e. the maximum number of white bits in the border. It is represented relative to the total number of bits in the marker.
b. **errorCorrectionRate (0.35)**: Each marker dictionary has a theoretical maximum number of bits that can be corrected (Dictionary.maxCorrectionBits). However, this value can be modified by the errorCorrectionRate parameter**.**

5) Corner Refinement

a. **cornerRefinementMethod() :** This parameter determines whether the corner subpixel process is performed or not and which method to use if it is being performed.
   i. CORNER_REFINE_NONE
   ii. CORNER_REFINE_SUBPIX
   iii. CORNER_REFINE_CONTOUR
   iv. CORNER_REFINE_APRILTAG

b. **cornerRefinementWinSize(5)** : This parameter determines the window size of the subpixel refinement process.

   High values can produce the effect that close image corners are included in the window region, so that the marker corner moves to a different and wrong location during the process. Furthermore it can affect performance.
c. **cornerRefinementMaxIterations(30) and cornerRefinementMinAccuracy(0.1):** These two parameters determine the stop criteria of the subpixel refinement process. The cornerRefinementMaxIterations indicates the maximum number of iterations and cornerRefinementMinAccuracy the minimum error value before stopping the process.
   If the number of iterations is too high, it can affect the performance. On the other hand, if it is too low, it can produce a poor subpixel refinement.

Current parameter values :

double	adaptiveThreshConstant
int	adaptiveThreshWinSizeMax 25
int	adaptiveThreshWinSizeMin 5
int	adaptiveThreshWinSizeStep 5
	adaptiveThreshConstant 15
float	aprilTagCriticalRad
int	aprilTagDeglitch

float	aprilTagMaxLineFitMse
int	aprilTagMaxNmaxima
int	aprilTagMinClusterPixels
int	aprilTagMinWhiteBlackDiff
float	aprilTagQuadDecimate
float	aprilTagQuadSigma
int	cornerRefinementMaxIterations
int	cornerRefinementMethod : CORNER_REFINE_SUBPIX
double	cornerRefinementMinAccuracy
int	cornerRefinementWinSize
bool	detectInvertedMarker
double	errorCorrectionRate
int	markerBorderBits
double	maxErroneousBitsInBorderRate
double	maxMarkerPerimeterRate
double	minCornerDistanceRate
int	minDistanceToBorder
double	minMarkerDistanceRate
double	minMarkerPerimeterRate 0.05 maxMarkerPerimeterRate 0.27
double	minOtsuStdDev
double	perspectiveRemoveIgnoredMarginPerCell
int	perspectiveRemovePixelPerCell
double	polygonalApproxAccuracyRate

Calculations;

OPENCV PARAMETER CALCULATIONS

- min marker perimeter = 0.03 Size: 640 x 480

∴ Perimeter = (0.03 x 640) x (0.03 x 480)

minperimeter = 19.2 x 14.4 pixels. = **19.2 px**

Polygon approx accuracy rate : 0.052

∴ minerror = Perimeter x 0.052

= 0.052 x (19.2 x 14.4) ppx

∴ min error = **1 x 0.75** = **1 px**

max marker parameter = 1

∴ max perimeter = **640 px**

⇒ max error = 640 x 0.052 = **33.28 px**

Polygon approx accuracy rate : 0.052

∴ max error = 33.28 px

minimum distance b/w 2 corners = min Perimeter * min corner distance rate.

= 19.2 x 0.05

= **1.92 ppx / 2** = **0.96 px**

minimum dist b/w corners of 2 markers = min. marker perimeter * min. marker distance rate.

= 19.2 px * 0.1

= 1.92 px