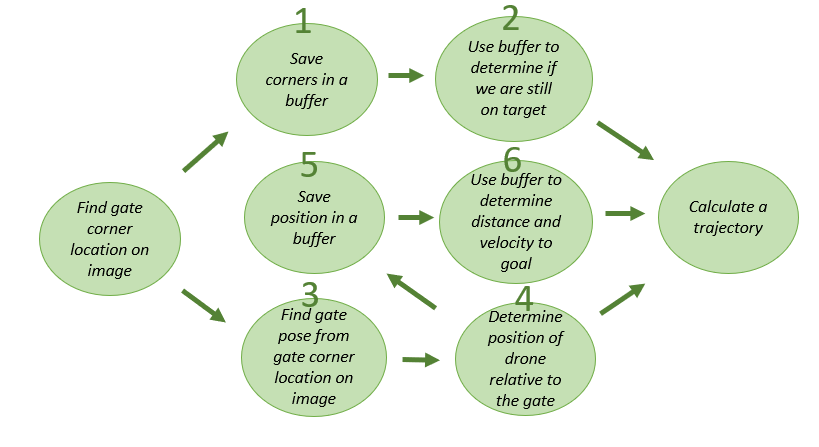
**How to make a robot know its position using only gate corners**

In this tutorial, we are given gate corners on a camera video feed and we use this video feed to determine the drone's position and velocity to the goal.



Finally we will be building our code on this foundation. As passing gates is our primary objective, let's see how we will process gate information.

**# Gate information**

The gate information updates everytime the corners are detected on the image. To do so, we first define gate parameters.

Please pay attention to these parameters: they govern the rest of the code.

typedef struct {

int target\_index; *//the number of the current gate*

VECTOR2 midpoint, lgm, corner[4], lgc[4];

*//using predefined VECTOR2 type ie. (x,y) coordinates*

*//midpoint of gate on screen*

*//lgm: last good midpoint, in case we lose it*

*//corner[4]: all 4 corners*

*//lgc[4]: all 4 last good corners in case we lose it﻿*

cv::Mat transform; *//to store 3D position of gate*

double length, distance;

*//length: the side length between 2 adjescent corners.*

bool found;

*//TRUE when corners are detected AND belong to the correct gate.*

*//FALSE otherwise.﻿*

bool passed;

*//TRUE when a gate length existed but now has disappeared: it's a way of saying we passed the gate!*

*//FALSE otherwise.*

} GATE\_INFO;

**# Ordering the corners for better processing**

std::vector<VECTOR2> clockwise\_corners;

if (n\_corners == 4) {

//std::cout << "n corners is " << n\_corners << std::endl;

GetClockwiseCorners(clockwise\_corners, gate\_info);

}

else if (n\_corners == 3) {

//std::cout << "n corners is " << n\_corners << std::endl;

FillAndGetClockwiseCorners(clockwise\_corners, gate\_info);

}

if (n\_corners >= 3) { // Last Good Midpoint

gate\_info->lgm = gate\_info->midpoint;

for (int i = 0; i < 4; i++)

gate\_info->lgc[i] = gate\_info->corner[i];

}

**# Reprojecting the corner coordinates to determine their 3D Pose in the environment.**

cv::Mat reprojection(768, 1024, CV\_8UC3, cv::Scalar(0,0,0));

double gateSize = 2.0;

std::vector<cv::Point3f> objPts;

objPts.push\_back(cv::Point3f(0,0,0));

objPts.push\_back(cv::Point3f(gateSize,0,0));

objPts.push\_back(cv::Point3f(0,gateSize,0));

objPts.push\_back(cv::Point3f(gateSize,gateSize,0));

std::vector<cv::Point2f> imgPts;

imgPts.push\_back(cv::Point2f(llx,lly));

imgPts.push\_back(cv::Point2f(lrx,lry));

imgPts.push\_back(cv::Point2f(ulx,uly));

imgPts.push\_back(cv::Point2f(urx,ury));

double fx = 548.4088134765625, cx = 512.0, fy = 548.4088134765625, cy = 384.0;

double camera\_mat\_vec[9] = { fx, 0.0, cx, 0.0, fy, cy, 0.0, 0.0, 1.0 };

cv::Mat cameraMatrix = cv::Mat(3, 3, cv::DataType<double>::type, camera\_mat\_vec);

cv::Mat distCoeffs = cv::Mat::zeros(4, 1, CV\_64FC1); *// vector of distortion coefficients*

cv::Mat rvec = cv::Mat::zeros(3, 1, CV\_64FC1); *// output rotation vector*

cv::Mat tvec = cv::Mat::zeros(3, 1, CV\_64FC1); *// output translation vector*

cv::solvePnP(objPts, imgPts, cameraMatrix, distCoeffs, rvec, tvec);

cv::Mat rotMatrix = cv::Mat::zeros(3, 3, CV\_64FC1); *// rotation matrix*

Rodrigues(rvec, rotMatrix);

std::vector<cv::Point2f> projPts;

std::vector<cv::Point3f> axisPts;

axisPts.push\_back(cv::Point3f(0, 0, 0)); *// Origin*

axisPts.push\_back(cv::Point3f(0, 0, 1)); *// Z-Axis*

axisPts.push\_back(cv::Point3f(0, 1, 0)); *// Y-Axis*

axisPts.push\_back(cv::Point3f(1, 0, 0)); *// X-Axis*

cv::projectPoints(axisPts, rvec, tvec, cameraMatrix, distCoeffs, projPts);

putText(reprojection, "UL", cv::Point(ulx, uly), cv::FONT\_HERSHEY\_PLAIN, 1.0, cv::Scalar(0, 0, 255), 1);

putText(reprojection, "UR", cv::Point(urx, ury), cv::FONT\_HERSHEY\_PLAIN, 1.0, cv::Scalar(0, 0, 255), 1);

putText(reprojection, "LR", cv::Point(lrx, lry), cv::FONT\_HERSHEY\_PLAIN, 1.0, cv::Scalar(0, 0, 255), 1);

putText(reprojection, "LL", cv::Point(llx, lly), cv::FONT\_HERSHEY\_PLAIN, 1.0, cv::Scalar(0, 0, 255), 1);

cv::line(reprojection, cv::Point(ulx, uly), cv::Point(urx, ury), cv::Scalar(255,255,255), 1);

cv::line(reprojection, cv::Point(urx, ury), cv::Point(lrx, lry), cv::Scalar(255,255,255), 1);

cv::line(reprojection, cv::Point(lrx, lry), cv::Point(llx, lly), cv::Scalar(255,255,255), 1);

cv::line(reprojection, cv::Point(llx, lly), cv::Point(ulx, uly), cv::Scalar(255,255,255), 1);

cv::line(reprojection, cv::Point(projPts[0].x, projPts[0].y), cv::Point(projPts[1].x, projPts[1].y), cv::Scalar(128,255,255), 2);

cv::line(reprojection, cv::Point(projPts[0].x, projPts[0].y), cv::Point(projPts[2].x, projPts[2].y), cv::Scalar(255,128,255), 2);

cv::line(reprojection, cv::Point(projPts[0].x, projPts[0].y), cv::Point(projPts[3].x, projPts[3].y), cv::Scalar(255,255,128), 2);

cv::Mat H, V;

cv::hconcat(rotMatrix, tvec, H);

double data[10] = { 0, 0, 0, 1 };

cv::Mat Z = cv::Mat(1, 4, CV\_64FC1, data);

cv::vconcat(H, Z, V);

cv::Mat trans = V.inv();

*// imshow("Reprojection", reprojection);cv::waitKey(1);*

gate\_info->transform = trans;

ii

double drone\_x = gate\_info->transform.at<double>(0,3); double drone\_y = gate\_info->transform.at<double>(1,3);

double drone\_z = gate\_info->transform.at<double>(2,3);

#Save gate pose to a buffer

std::array<double, 3> item;

item[0]= x;

item[1]= y;

item[2]= z;

if(dronePoseWithRespectToGateVector.size() >= 10){

dronePoseWithRespectToGateVector.erase(dronePoseWithRespectToGateVector.begin(), dronePoseWithRespectToGateVector.begin() + 1);

}

if(droneDistWithRespectToGateVector.size() >= 10){

droneDistWithRespectToGateVector.erase(droneDistWithRespectToGateVector.begin(), droneDistWithRespectToGateVector.begin() + 1);

}

double currDist = lastGoodDistance;

char pos\_buffer [50];

sprintf(pos\_buffer, " Distance %f ", currDist);

*// drone distance history vector*

droneDistWithRespectToGateVector.push\_back(currDist);

*// Pose history vector*

dronePoseWithRespectToGateVector.push\_back(item);

**#Gate length calculation**

The gate length is an approximation of the length of a side of the gate. Because it is determined from the camera information, the fastest way to calculate it is by measuring the distance between two sides.

gate\_info->length = 0;

int n\_sides = 0;

if (gate\_info->corner[0].x && gate\_info->corner[3].x) {

gate\_info->length = distance(gate\_info->corner[0], gate\_info->corner[3]);

n\_sides++;

}

if (gate\_info->corner[1].x && gate\_info->corner[2].x) {

gate\_info->length += distance(gate\_info->corner[1], gate\_info->corner[2]);

n\_sides++;

}

if ((n\_sides == 0) && gate\_info->corner[2].x && gate\_info->corner[3].x) {

*// Only use bottom edge if neither side is seen.*

gate\_info->length += distance(gate\_info->corner[2], gate\_info->corner[3]);

n\_sides++;

}

gate\_info->length = n\_sides ? gate\_info->length / n\_sides : -1;

gate\_info->distance = 700 / gate\_info->length;

ooo

fdsfd