

# Planning based on Voronoi graph and visibility graph

Yuhao Zhang

May 25, 2018

## 1 Introduction

The project is an implementation of both Voronoi diagram & visibility graph planning on Sunfounder's PiCar-V platform and a webcam attached to it is used. With the help of QR codes as landmarks, the webcam is turned into a distance-bearing sensor.

## 2 Visibility graph

Visibility graph is a way to describe obstacles on the 2-D plane of the robot motion. Each node in the graph represents a point on the plane, and edges connecting vertices indicate that there is a visible connection between these two vertices. In our case of mobile robot on 2-D plane, the nodes will simply be the vertices of the obstacles, if they are approximated as simply polygons. And the collection of edges become the viable roadmap of this plane.

To utilize this notion, we can use a set of vertices of the polygons to represent the obstacles, a set of points to represent the waypoints and a single point to represent the robot. The edges of the polygons are obstacles. Then in order to plan the path, a shortest path algorithm such as Dijkstra's algorithm can be applied to this graph.

Planner based on visibility graph has optimal result, i.e. strictly shortest path on the plane, yet it is prone to collision as the path is not safe, for it needs to drive at the vicinity of the obstacles.

## 3 Voronoi graph

An alternative option to visibility graph is the Voronoi graph, which is technically a partitioning of the plane based on the pre-specified seeds on the plane. The plane is partitioned into regions and each region has its corresponding seed, meaning that the seed is closer to all points inside the region than any other seed. A vertex is then defined as the point that

has three or more seeds that are equally distant from it. An edge is a collection of points that have exactly two seeds that are equally distant from. Notice that there may exist edges that are connecting points at infinity instead of finite vertices.

To apply Voronoi graph for planning. We use a set of points of the outline of the obstacles to approximate them. These points will server as the seeds of the Voronoi graph. Further an finite box is drawn as the finite representation of the 2-D plane. This box will automatically eliminate all edges connecting infinity.

Then the generated Voronoi graph automatically becomes the roadmap of the plan. To actually plan a path, given staring point and the destination point, the nearest vertices or nearest points on any edge of the Voronoi graph are obtained as the points to enter/exit the roadmap. The robot will first be driven to the enter point of the Voronoi graph, then a shortest path algorithm such as Dijkstra’s algorithm can be applied to this graph to find the path to the exit point. Finally the robot will exit from the exit point and drive to the destination.

Voronoi graph constructed this way will yield the roadmap that is most far away from any obstacle on the plane, therefore makes it the safest route and is mostly useful when the optimal path is not required.

## 4 Localization

To further improve the performance of the planning, localization of the mobile car can be incorporated. One major problem for Sunfounder’s PiCar-V platform is the lack of a distance-bearing sensor that would make localization with landmarks much easier. However, with the help of known-sized QR tags as 3D object, it is plausible to convert a plain webcam into a distance-bearing sensor.

### 4.1 Distance and Pose estimation using PnP

With a simple pinhole model the scene view  $s$  of a camera is formulated as:

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix},$$

where  $(X, Y, Z)$  is the coordinate of the object point in world reference frame,  $(u, v)$  is the coordinates of the projected object on the view. The first factor on the right hand side is the camera matrix which depicts the intrinsic properties of the camera, while the second factor

is the rotation-translation matrix that transforms from world reference frame to the camera frame.

The rotation-translation matrix  $R - T$  is solvable by first, detecting the QR code using library such as `zbar`. Second, finding the corner points of it and associate with their projections on the view. Once  $R - T$  is obtained, one can solve directly for its inverse, as the inverse serves directly for pose estimation of the camera.

Given  $R$  which is the rotation matrix, the inverse of it is denoted as  $R^{-1} = R^T$ . Then the coordinates of the camera in world reference frame are obtained as

$$(x, y, z) = -R^T V.$$

And the distance between the camera and the object:

$$d = \sqrt{(X - x)^2 + (Y - y)^2 + Z - z)^2}.$$

Furthermore, the Euler angles representation can also be acquired easily form  $R$ . As for car-like mobile robots, the motion is usually confined to 2-D plane. Consider such a plane is depicted is a right-hand coordinate system as  $x - z$  plane and axis  $y$  is pointing to the ground. Then only Euler angle  $\theta_y$  is of interest. Then the problem can be largely simplified. For a landmark  $M(X, Y, \gamma)$  in world reference frame, the current pose of the camera is

$$(X - d \cos(\theta_y), Y + d \sin(\theta_y), \pi/2 - \theta_y + \gamma).$$

## 4.2 Bearing

The bearing of the object with respect to the camera can be calculated directly using the rotation-translation matrix  $R - T$ . But in the experiments such calculation turns out to be quite inaccurate and is not sufficient for real application. Therefore in the rest of the paper the bearing is obtained as

$$(c_x - c_c) \times \frac{d}{f_x},$$

where  $c_x$  is the  $x$  principle point as shown in the camera matrix,  $c_c$  is the center of the QR tag in camera frame and  $f_x$  is the  $x$ -direction focal length of the camera.

With this pose estimation, one can combine a control algorithm with feedback from QR codes detected by camera.

A picture showing the performance of such model is plotted as Fig. (1).

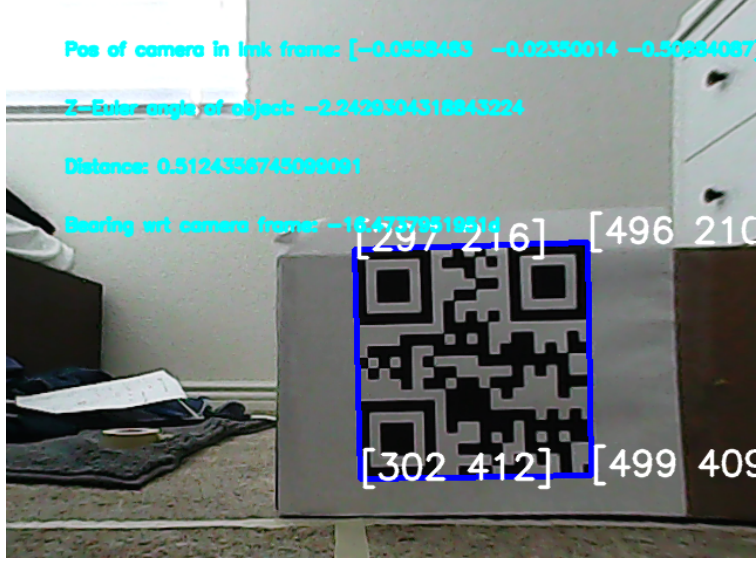


Figure 1: The QR code used as a landmark.

## 5 Kinematics and control law

The kinematics of the car is orthogonal to the current Ackerman model can be used to describe the kinematics of a car-like robot:

$$\frac{dx}{dt} = v \cos \theta,$$

$$\frac{dy}{dt} = v \sin \theta,$$

$$\frac{d\theta}{dt} = \frac{v}{L} \tan \gamma,$$

where  $(x, y)$  is the position of the middle point of the back wheel axis of the robot in world reference frame.  $\theta$  is the angle of pose of the robot. The steering wheel angle is  $\gamma$  and the velocity of the back wheel is  $v$ .  $L$  is the length of the vehicle or wheel base.

These equations can be re-written as:

$$\begin{pmatrix} \frac{dx}{dt} \\ \frac{d\theta}{dt} \end{pmatrix} = \begin{pmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix}.$$

With the initial position-pose  $(x, y, \theta)$  and the goal  $(x^*, y^*, \theta^*)$ , it is more convenient to write the equations in polar system via a transformation:

$$\rho = \sqrt{\Delta_x^2 + \Delta_y^2},$$

$$\alpha = \arctan \frac{\Delta_y}{\Delta_x} - \theta,$$

$$\beta = -\theta - \alpha + \theta^*.$$

The linear control law for  $-\pi/2 < \alpha \leq \pi/2$ , i.e. the waypoint is in front of the vehicle is:

$$v = k_\rho \rho,$$

$$\omega = k_\alpha \alpha + k_\beta \beta,$$

where  $k_\rho, k_\alpha, k_\beta$  are arbitrary coefficients that satisfies  $k_\rho > 0, k_\beta < 0, k_\alpha - k_\rho > 0$ .

The control law for the cases where the waypoint is behind the vehicle is the same as above, but with transformed angles:

$$\alpha' = -\pi - \beta,$$

$$\beta' = -\pi - \alpha,$$

and  $v' = -v$ .

## 6 implementation and setup

The visibility graph planner purposed in Sec.(2) is implemented in `visibility.ipynb` based on a visibility graph library `pyvisgraph` using Lee's visibility graph algorithm, the path finding algorithm is the Dijkstra's algorithm. The actual setup of the surroundings and the planned path is plotted as Fig.(2).

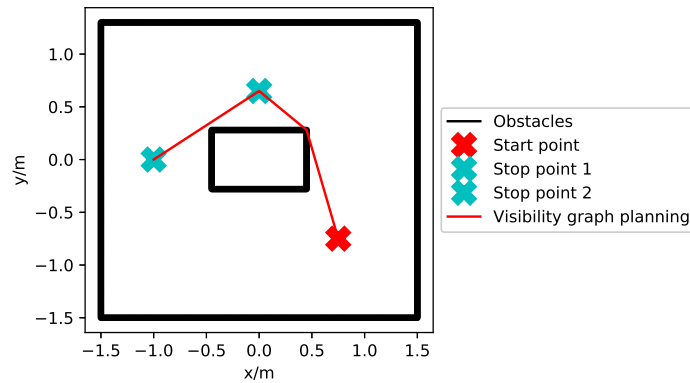


Figure 2: The actual setup of the surroundings and the planned path by visibility graph planner

The Voronoi graph planner purposed in Sec.(3) is implemented in `Voronoi.ipynb`, the obstacle is represented by a set of points. The Voronoi graph is generated using `scipy.spatial.Voronoi` and the graph is stored and managed using `networkx`. The path finding algorithm is also the Dijkstra's algorithm. The actual setup of the surroundings and the planned path is plotted as Fig.(3).

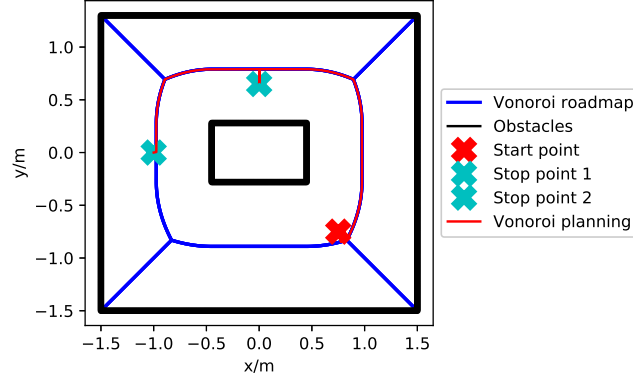


Figure 3: The actual setup of the surroundings and the planned path by Voronoi graph planner

As of the controller of the robot, one can apply a time-sample linear controller together with an EKF based localization system with the QR tags. Yet for the sake of simplicity, in this project a straight line and constant turning controller is used.

## 7 Experiment and conclusion

The actual performance of the two planners are shown in Tab.(1). It can be seen that the

Table 1: Experiment results			
Way points	Planner	Error (m)	Drive time(s)
Strat point	Visibility graph	—	-
	Voronoi graph	—	-
Stop point 1	Visibility graph	$0.1 \pm 0.05$	6.45
	Voronoi graph	$0.1 \pm 0.05$	7.50
Stop point 2	Visibility graph	$0.15 \pm 0.05$	3.00
	Voronoi graph	$0.2 \pm 0.05$	4.5

two planner performs no difference in terms of the driving accuracy. But the route provided by visibility graph planner is much faster than the path produced by Voronoi graph planner. As the visibility graph planner tends to give dangerous yet optimal path, while the Voronoi graph planner gives safe yet conservative path, in experiment the robot runs into the obstacle in about 30% of all runs, and the Voronoi graph planner never drives the robot into collision. The runs that collides with the obstacle are not taken into account in statistics.

The conclusion is visibility graph is for optimal path but poses higher risk of collision, and Voronoi graph planner yields reliable yet slow route. It depends on the task requirement to decide which one is best suitable.