FINAL PROJECT REPORT

MAE 5810

Pinhole Camera Tracking and Path Planning Based on Two-wheeled Vehicle Differential Steering Model

Yang Jiao

NetID: yj286

10 Dec 2021

ABSTRACT

This article introduced the establishment of differential steering model for two-wheeled vehicles, and realized the trajectory tracking and path planning for two-wheeled vehicles by combining the pinhole camera model constructed in the mid-term project. At the same time, the A*-based heuristic search algorithm was explained considering the possible environmental obstacles in the actual application scenarios, and the simulation results were illustrated.

1. Differential steering model for two-wheeled vehicles

There are two models in steering control, one is the bicycle model and the other is the differential model. We assume that a two-wheeled vehicle is in a steering state as shown below.

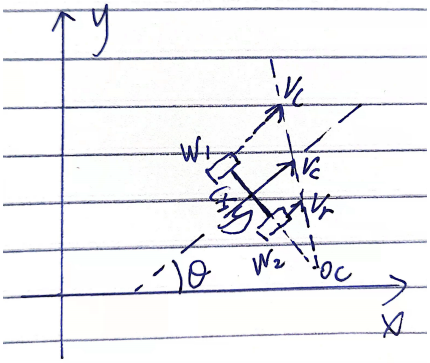

Figure 1: Two-wheeled vehicle in a steering state

Assume that the angular velocity of the motor output is ϕl and ϕr, the size of the wheel is r, calculate the speed on each side of the wheels:

$$v_l = \frac{\phi_l \cos(\theta)}{2}$$

$$v_r = \frac{\phi_r \cos(\theta)}{2}$$

Now the speed of the central point is $v_c = \frac{v_r + v_l}{2}$

The spacing between the left and right wheels is l, and the distance from the turning radius center O_c to c is R. When turning $w_l = w_r = w_c$, because the vehicle is coaxial when turning. These yields:

$$l = \frac{v_r}{w_r} - \frac{v_l}{w_l}$$
$$w_c = \frac{v_r - v_l}{l}$$

Calculate R:

$$R = \frac{v_c}{w_c} = \frac{l(v_r + v_l)}{2(v_r - v_l)}$$

Combining the geometrical relations in the figure, using a similar derivation as in the previous homework assignment deduces that:

$$x' = v_c \cos\,\theta = \frac{v_r + v_l}{2} \cos\,\theta$$
$$y' = v_c \sin\,\theta = \frac{v_r + v_l}{2} \sin\,\theta$$
$$\theta' = w_c = \frac{v_r - v_l}{l}$$

To be able to simulate this model in MATLAB, consider discretizing the above model to obtain (where T is the sample time):

$$x(t+1) = x(t) + v_c \cos\,\theta \times T$$
$$y(t+1) = y(t) + v_c \sin\,\theta \times T$$
$$\theta(t+1) = \text{ theta } (t) + w_c \times T$$

Consider that in the actual situation, the speed of the left and right motors is mainly controlled to control the steering of the own vehicle, so that v_r and v_l can be inversely solved from the above equation.

$$v_r = v_c + \frac{w_c l}{2}$$
$$v_l = v_c - \frac{w_c l}{2}$$

The above model was implemented in MATLAB. The pinhole camera model will be introduced in section 2. With tracking and planning introduced in section 3.

2. Pinhole Camera Model

Consider a PT camera in a 3D space, the camera is located at $X\ (x_I, y_I, z_I)$, set a to be the width of the image sensor and b to be the height. λ denotes the distance from the target plane to the virtual vertices plane. Define the pan-tilt angle as $\psi \in [0,2\pi)$ and $\phi \in [\pi/2, \pi]$.

To perform target tracking, first assume a non-moving target located at $(xtar, ytar, ztar)$. On the virtual plane, the target can be observed in the rectangle formed with:

$$(\mathbf{q}_1)_V = \begin{bmatrix} \frac{a}{2} \\ \frac{b}{2} \end{bmatrix}, (\mathbf{q}_2)_V = \begin{bmatrix} \frac{a}{2} \\ -\frac{b}{2} \end{bmatrix}, (\mathbf{q}_3)_V = \begin{bmatrix} -\frac{a}{2} \\ -\frac{b}{2} \end{bmatrix}, (\mathbf{q}_4)_V = \begin{bmatrix} -\frac{a}{2} \\ \frac{b}{2} \end{bmatrix}, \tag{1}$$

The virtual plane is λ away from the target platform, so the position relative to the camera-based frame is:

$$(\mathbf{q}_\ell)_B = [(\mathbf{q}_\ell)_V^T \quad \lambda]^T, \ell = 1, \dots, 4 \tag{2}$$

Perform the pan and tilt Euler-angle rotation to represent the rotation through rotation matrices which also include the axis that is not changed during the movement:

$$\mathbf{H}_\psi \triangleq \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{H}_\phi \triangleq \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{3}$$

For the vectors in (2), use the transformation matrices to first transform into inertial and extend the vector to reach the target plane by adjusting the factor. Sensor physical bounds in the inertial frame can be calculated using the inverse transformation to the transpose of rotation matrices in the following way:

$$\begin{aligned} (\mathbf{q}_\ell)_I &= \left(\mathbf{H}_\phi^T \mathbf{H}_\psi^T\right)^{-1} (\mathbf{q}_\ell)_B \\ &= \left[\left(\mathbf{H}_\psi \mathbf{H}_\phi\right)^{-1}\right]^T (\mathbf{q}_\ell)_B \\ &= \mathbf{H}_\psi \mathbf{H}_\phi (\mathbf{q}_\ell)_B, \ell = 1, \dots, 4 \end{aligned} \tag{4}$$

From equation (4), we can obtain the inverse transformation of $\mathbf{q}_\ell$, then we add a factor $\rho_\ell \in \mathbb{R}_+$ to adjust the vector to reach the 2D target plane which is described in the following equation:

$$(\mathbf{q}'_\ell)_I = \rho_\ell (\mathbf{q}_\ell)_I, \ell = 1, \dots, 4 \tag{5}$$

To reach the 2D target plane: $\mathbf{q}'_{\ell_{(3)}} = -\mathbf{x}_{(3)}, \ell = 1, \dots, 4$

Calculate $(\mathbf{q}'_\ell)_I$ using the proposed factor on the FOV bound and substitute the pan

and tilt angle into the equation to represent the factor as:

$$\rho_{1,4} = \frac{-\mathbf{x}_{(3)}}{\left(\frac{b}{2}\sin\phi + \lambda\cos\phi\right)} \text{ and } \rho_{2,3} = \frac{-\mathbf{x}_{(3)}}{\left(-\frac{b}{2}\sin\phi + \lambda\cos\phi\right)} \quad (6)$$

Substitute (6) into equation (5) to obtain the FOV vertices

$$\xi_1 = \boldsymbol{X} + \mathbf{q}_1'$$
$$\xi_2 = \boldsymbol{X} + \mathbf{q}_2'$$
$$\xi_3 = \boldsymbol{X} + \mathbf{q}_3'$$
$$\xi_4 = \boldsymbol{X} + \mathbf{q}_4'$$

To visualize the camera FOV and the target we need to implement the whole process in MATLAB in the following section and add control algorithms to facilitate target tracking.

3. Distance /Heading angle Control and Tracking

First, the coordinates of the center point of the FOV were calculated on the plane where the target is located:

```
xcenter=[v1(:, 1)+ v2(:, 1)+v3(:, 1)+v4(:, 1)]/4;
ycenter=[v1(:, 2)+v2(:, 2)+v3(:, 2)+v4(:, 2)]/4;
```

To control the camera model and finally achieve the trajectory tracking of the target, the error values of the pan and tilt angles corresponding to the target and the center of bottom FOV were illustrated in the 3-axis figure below:

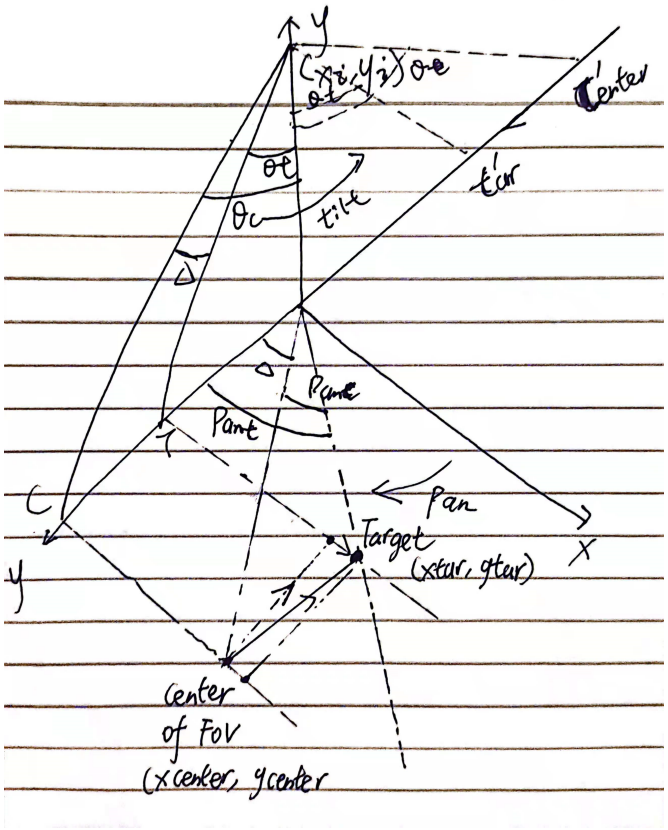

Figure 2: 3-axis illustration of pan and tilt error values

From the figure, we can see that the relative position of the center point and the target can be represented by pan angle error and tilt angle error, and the angle error can be controlled by introducing negative feedback, but there are still some problems.

The quadrant where center and target are located has a key influence on the determination of the symbol. To facilitate the analysis, the angle changing planes of pan and tilt are abstracted and the positive and negative sign cases of tilt angles in the quadrants on the bottom plane are first analyzed as below:

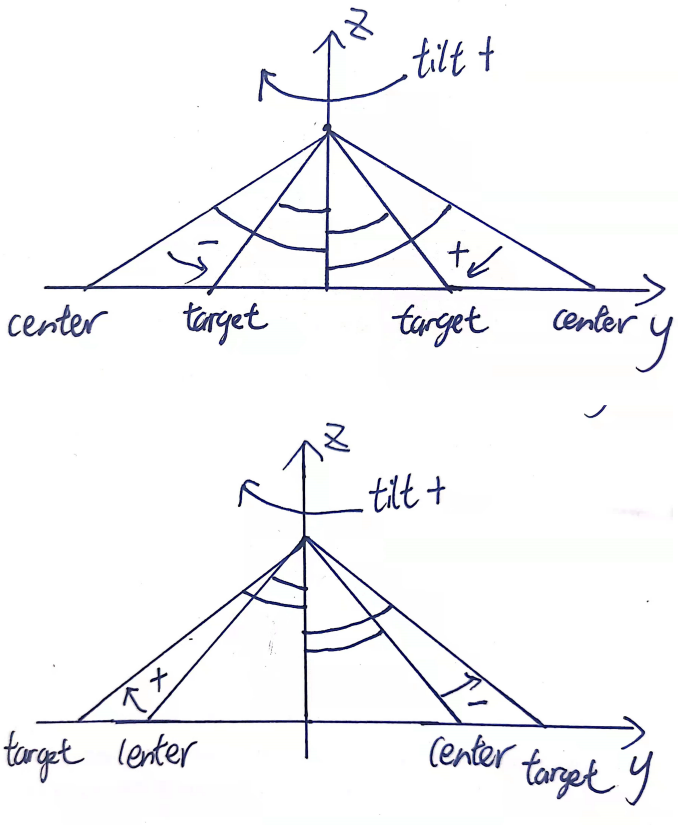

Figure 3: Tilt error analysis of judgment conditions

$$errortilt = atan(\frac{lenctilt}{z_i}) - atan(\frac{lenttilt}{z_i});$$

If choose to set the delta of tilt angles to be the tilt angle of the center minus the tilt angles of target, then a symbol judgment logic can be derived as if y>0, the error should be positive, and when y<0, the error should add another minus symbol to ensure negative feedback is established.

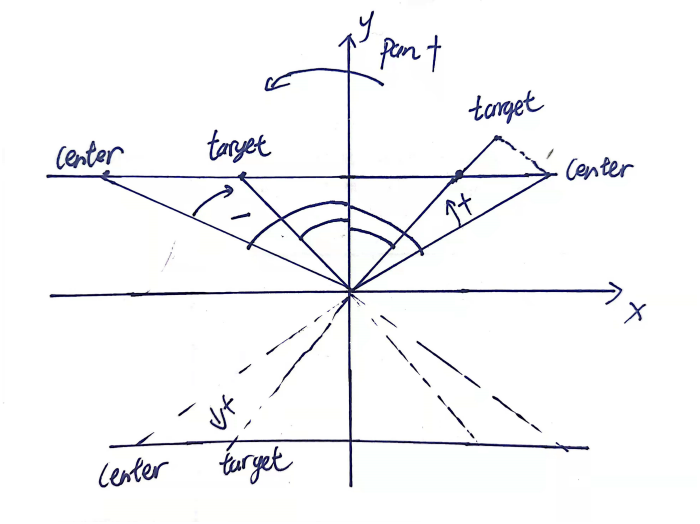

Figure 4: Pan error analysis of judgment conditions

$$errorpan = atan(abs(\frac{xcenter}{ycenter})) - atan(abs(\frac{xtar}{ytar}));$$

If choose to set the delta of tilt angles to be the tilt angle of the center minus the tilt angles of target, then a symbol judgment logic can be derived as if x*y<0, the error should be positive, and when x*y>0, the error should add another minus symbol to ensure negative feedback is established.

Set the initial heading angle to be $\frac{pi}{2}$, then

$$theta_{goal} = \text{atan}\,(\frac{y_{goal} - ytrac}{x_{goal} - xtrac});$$

$$theta_{error} = theta - theta_{goal};$$

$$error_{sum} = error_{sum} + theta_{error}$$

Then the control input of heading angle is:

$$w = -k_p * (theta_{error}) + k_i * error_{sum} + k_d * (theta_{error} - pre_{error});$$

For the distance control

$$dist = \left((xtrac - x_{goal})^2 + (ytrac - y_{goal})^2\right)^{\frac{1}{2}};$$

Then the speed of the vehicle is:

$$v = k2 * dist;$$

Where the left and right wheel's speed are:

$$v_r = v + w * \frac{L}{2}$$

$$v_l = v - w * \frac{L}{2}$$

The control dynamic model as established in section 1 is:

$$v = (vl + vr)/2$$

$$w = (vr - vl)/L$$

$$xtrac = xtrac + v * cos(theta) * T$$

$$ytrac = ytrac + v * sin(theta) * T$$

$$theta = theta + w * T$$

This completes the tracking control of the two-wheeled vehicle model derived in Section 1.

4. A-Based Heuristic Search Algorithm for path planning*

In practice, there may be some obstacles between the start and end points, and how to plan the path while avoiding the obstacles is the issue to be discussed in this section. This section uses the A* algorithm to implement path planning. A* algorithm is a heuristic search algorithm, that is, heuristic search rules are established in the search process, to measure the distance relationship between the real-time search position and the target position, so that the search direction is given priority towards the direction of the position where the target point is located, and finally achieve the effect of improving the search efficiency.

Suppose there is an obstacle as shown in the figure below:

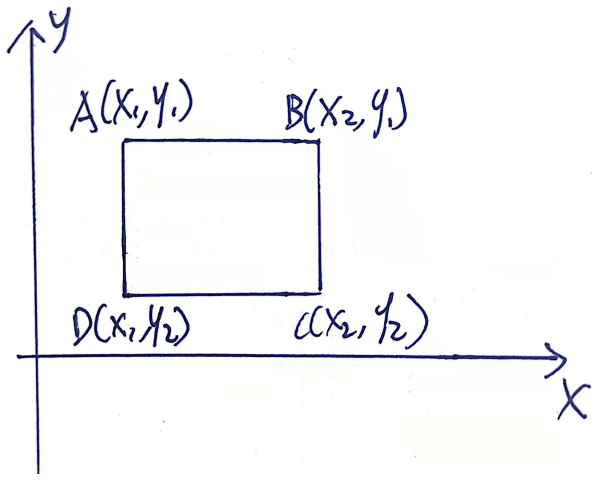

Figure 5: Obstacle on the target plane

The heuristic function for estimating the distance between points A and C must be transformed according to the heuristic information as:

$$H(n) = \bar{AB} + \bar{BC} = |x_1 - x_2| + |y_1 - y_2|$$

Evaluation function

$$F(n) = G(n) + H(n)$$

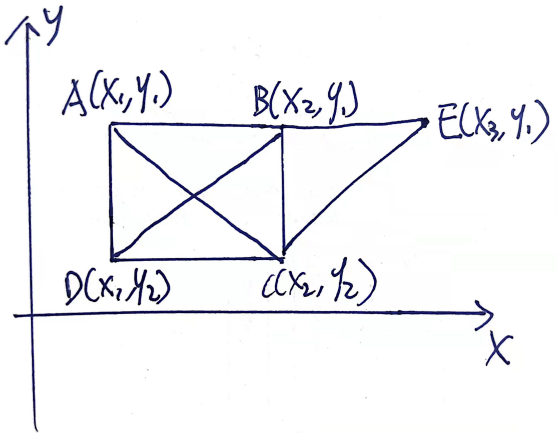

Figure 5: Paths from A to E

$G(n)$ is the actual value of the cost already paid from the starting vertex to the current vertex.

As shown in Figure 5, assuming that B is chosen as the current vertex, $G(n)$ represents the distance traveled from vertex A to vertex B,

$$G(\mathrm{n}) = \bar{AB} = |\mathrm{x}_1 - \mathrm{x}_2|;$$

$\mathrm{H(n)}$ is an estimate of the cost to be paid from the current vertex to the end point.

$$\mathrm{H(n)} = \bar{BE} = \sqrt{(x_2 - x_3)^2 + (y_1 - y_2)^2}$$

$$\mathrm{F(n)} = \mathrm{G(n)} + \mathrm{H(n)} = \bar{AB} + \bar{BE} = |\mathrm{x}_1 - \mathrm{x}_2| + \sqrt{(x_2 - x_3)^2 + (y_1 - y_2)^2}$$

The size of the obtained $\mathrm{F(n)}$ value indicates the importance of the current vertex. The smaller the F(n), the smaller the cost (shorter the distance) to reach the end point via that vertex, the stronger the importance of that vertex.

(1) Two tables named OPEN and CLOSED are defined; OPEN is used to store the unexamined vertices and CLOSED table is used to store the vertices that have been examined.

(2) Assume that there are n vertices in the graph, choose vertex Vs as the starting point, vertex Vd as the target point, vertex Vi: $(i <= n)$ for the remaining locations, put the starting point Vs into the OPEN table, and the CLOSED table is initialized to empty.

(3) Determine whether the OPEN table is empty, if empty means the search process failed.

(4) If not empty, move the first vertex Vi in the OPEN table into the CLOSED table.

(5) Determine whether Vi is the target vertex Vd, if yes, it means the search is successful and the algorithm is finished.

(6) If not then extend the search for the subvertex Vj (j<=n, j≠i) of Vi and calculate its F(Vj) value to determine whether Vj exists in the OPEN table or CLOSED table.

If Vj does not exist in the OPEN table and CLOSED table, choose to store it in the OPEN table and set the pointer to the parent vertex Vi. If Vj already exists in the OPEN table, update the F(Vj) value in the OPEN table, i.e., take the new smaller F(Vj) value to replace the old larger F(Vj) value in the OPEN table, and set the pointer to the parent Vi. If Vj already exists in CLOSED table or obstacle point,

ignore this vertex and return to step (6). Sort the vertices in the OPEN table in ascending order by the size of F(Vj) value, and then skip to step (3).

The search process of the A-star algorithm is implemented in the code as follows.

1’ Calculate the costs of all the blocks that are immediately next to the present one.

2’ Add them to valid list

3’ Find block with minimum total cost from valid list, make it parent for next iteration and make it unavailable for next minimum total cost comparison

4’ Put that in in_valid list so it won’t be visited again

5’ Keep doing this until the block with minimum cost is not target block.

CONCLUSION

This article mainly focused on the mathematical part of the final project. In the first part, introduced the establishment of differential steering model for two-wheeled vehicles, and realized the trajectory tracking and path planning for two-wheeled vehicles by combining the pinhole camera model constructed in the mid-term project. At the same time, the A*-based heuristic search algorithm was used considering the possible environmental obstacles in the actual application scenarios, and the simulation results were illustrated following the explained algorithm.