CSE 276A HW2

Tingyu Shi(A59023729)
Jiajun Li(A16635772)

December 26, 2024

1 Video URL

URL: https://youtu.be/0wingcaVLas

2 Closed Loop Controller

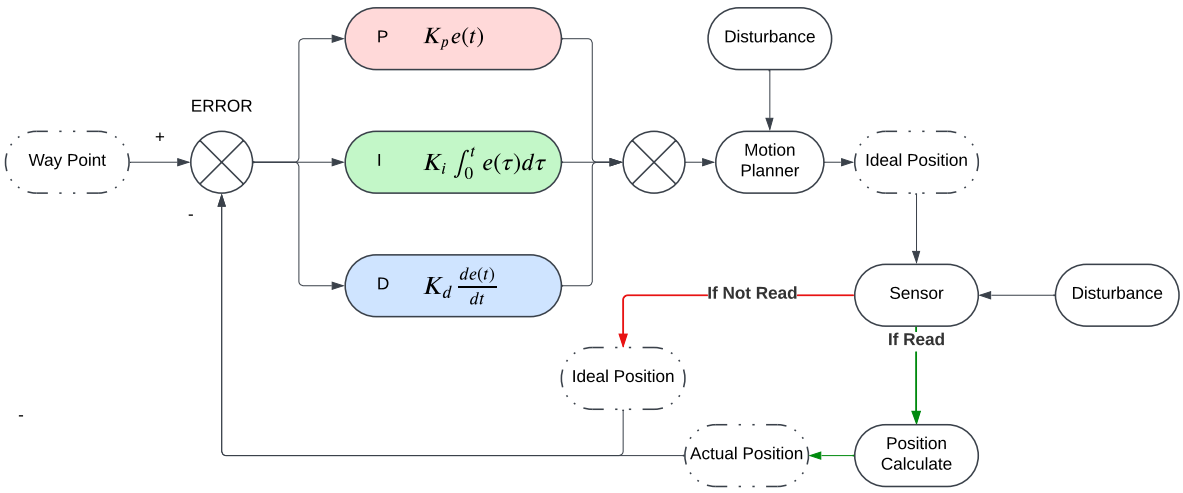

Figure 1: Closed Loop Controller

Figure 1 illustrates the control logic behind our car's movement. We begin by setting way points, which are specific locations that the car needs to pass through. The system calculates the error between each way point and the car's current position, feeding this into the controller. We use a PID controller, which effectively manages the error through proportional, integral, and derivative terms. The controller's output then informs the motion planner, which determines the necessary actions for the car. However, motor inaccuracies can introduce disturbances, potentially affecting the car's position.

To correct for this, a camera sensor reads an AprilTag to assess the car's actual position. If the camera detects the tag, it uses this to accurately locate the car; otherwise, it relies on the estimated position provided by the controller. Once the error falls below a set threshold, the way

point is updated to guide the car toward the next point. This loop repeats until all way points are completed.

3 Landmark Description

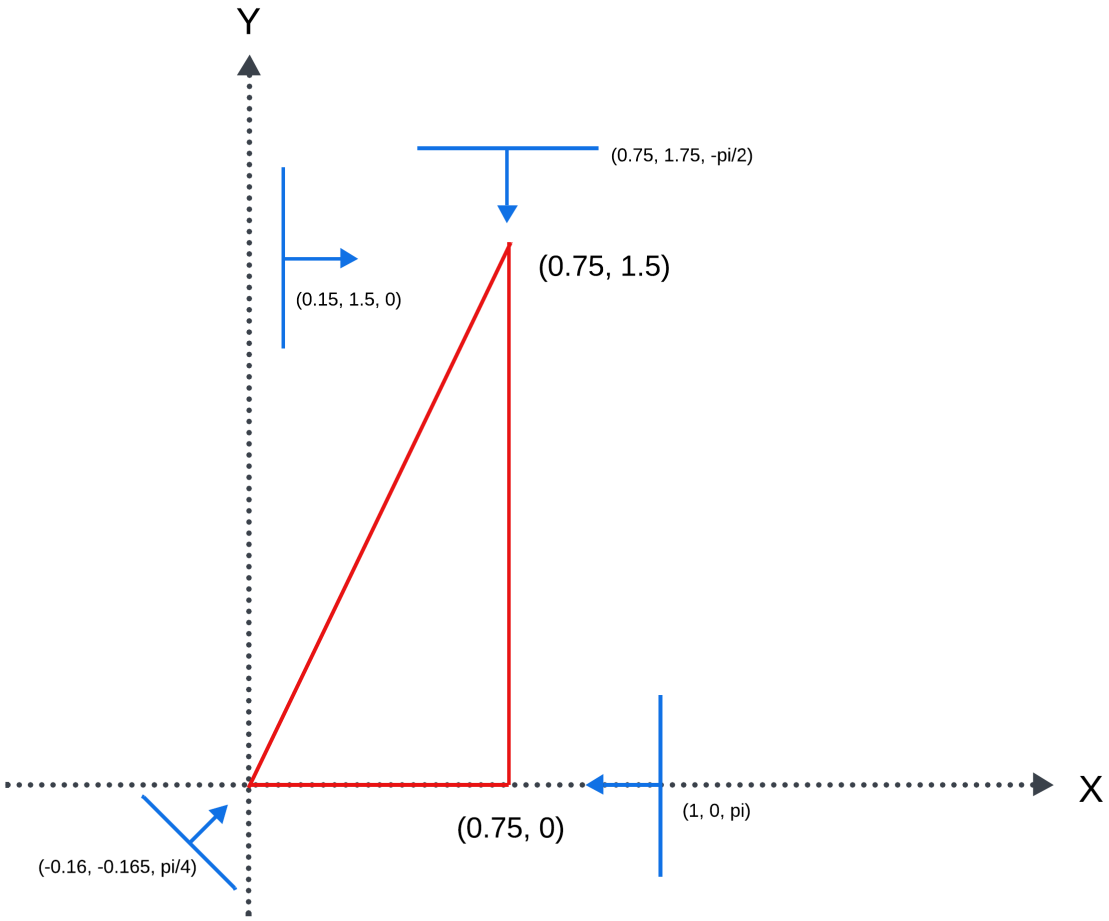

Figure 2: Landmarks – Unit Distance = 0.75m

Figure 2 shows the positions of landmarks. The red line represents the path, and the blue lines represent the April tags. The blue arrows point the direction that April tags face. We positioned April tags in this way so that the camera can capture at least one April tag while running. By doing this, the car can better estimate its current position.

4 Car Position Estimation

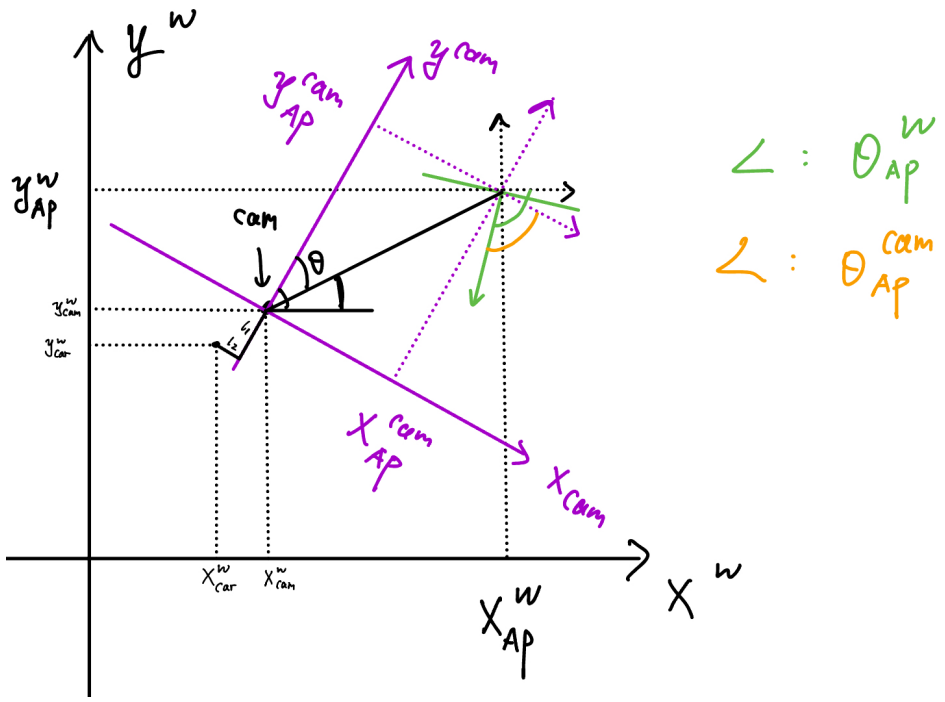

Figure 3: Coordinate System

Figure 3 describes the coordinate system we use. x^W and y^W represent the world coordinate system, x^{cam} and y^{cam} represent the camera coordinate system. $(x_{AP}^{cam}, y_{AP}^{cam}, \theta_{AP}^{cam})$ represent April tag's coordinate in camera system, $(x_{AP}^{W}, y_{AP}^{W}, \theta_{AP}^{W})$ represent April tag's coordinate in world system. During the experimentation, we found that there are inaccuracies of April tag detection algorithm, therefore, we believe it's fine to omit the translation between camera center and car center. The coordinate system is not the same as what's defined in April tag detection node, we did certain changes so that it fits our own coordinate system.

4.1 Using Translation and Rotation Matrix

The most efficient way to calculate the car's position in world coordinates is by using rotation and translation matrices. The camera provides us with the x, y values and the axis-angle representation of the AprilTag within the camera's coordinate system. Additionally, we know the x, y coordinates and the angle of the AprilTag in the world coordinate system. The transformation matrix is generally defined as follows:

$$\begin{bmatrix} cos(\theta) & -sin(\theta) & d_x \\ sin(\theta) & cos(\theta) & d_y \\ 0 & 0 & 1 \end{bmatrix}$$

(PS:We do not care about z-axies, because the car cannot fly)

To determine the car's position in world coordinates, we need to construct two transformation matrices. The first matrix transforms the car's coordinates to the AprilTag's coordinates, and the second transforms the AprilTag's coordinates to the world coordinates. Before constructing these

matrices, we must convert the axis-angle representation of the AprilTag's position from the camera's perspective to a pitch, yaw, and roll system. This is necessary because the axis-angle representation can be inconsistent in direction: both positive and negative axes can represent the same rotation. By contrast, the pitch, yaw, and roll system ensures a consistent z-axis direction. In this case, only the yaw value is needed due to the specific placement of our AprilTag.

Given an axis-angle representation with rotation axis (x, y, z) and angle θ, we can derive the rotation matrix R:

$$R = \begin{bmatrix} \cos\theta + x^2(1-\cos\theta) & xy(1-\cos\theta) - z\sin\theta & xz(1-\cos\theta) + y\sin\theta \\ yx(1-\cos\theta) + z\sin\theta & \cos\theta + y^2(1-\cos\theta) & yz(1-\cos\theta) - x\sin\theta \\ zx(1-\cos\theta) - y\sin\theta & zy(1-\cos\theta) + x\sin\theta & \cos\theta + z^2(1-\cos\theta) \end{bmatrix}$$

From R, the pitch (ϕ), yaw (ψ), and roll (γ) can be extracted as:

$$\phi = \arcsin(-R_{31})$$

$$\psi = \arctan 2(R_{21}, R_{11})$$

$$\gamma = \arctan 2(R_{32}, R_{33})$$

With this setup, we can build the matrices. For the first matrix (camera to tag coordinates), we use the yaw value as the rotation angle and the camera's x, y values as the translation components. This matrix represents how a vector in the tag's coordinate system appears in the camera's coordinates:

$$\begin{bmatrix} cos(\theta_{yaw}) & -sin(\theta_{yaw}) & camera\_read\_x \\ sin(\theta_{yaw}) & cos(\theta_{yaw}) & camera\_read\_y \\ 0 & 0 & 1 \end{bmatrix}$$

Since we need the reverse transformation, we take the inverse of this matrix. The inverse of a transformation matrix involves separately handling the rotation and translation components. The inverse of a transformation matrix T, which combines rotation and translation, is given by:

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

The inverse T^{-1} is then:

$$T^{-1} = \begin{bmatrix} R^T & -R^T t \\ 0 & 1 \end{bmatrix}$$

Here, R^T is the transpose of R, which is the inverse of the rotation matrix, and $-R^T t$ gives the negated transformed translation.

The second matrix transforms coordinates from the tag's system to the world coordinate system. Here, we use the x, y coordinates of the AprilTag in the world system, along with its placement angle. This matrix represents how a vector in the tag's coordinate system maps to the world coordinates:

$$\begin{bmatrix} cos(\theta_{tag\_angle}) & -sin(\theta_{tag\_angle}) & tag\_world\_x \\ sin(\theta_{tag\_angle}) & cos(\theta_{tag\_angle}) & tag\_world\_y \\ 0 & 0 & 1 \end{bmatrix}$$

With both matrices ready, we combine them by matrix-multiplying the second matrix with the first. The resulting matrix enables us to input the x, y values in the camera's coordinate system and output them in world coordinates. Finally, to calculate the car's angle in the world coordinate system, we use the formula $90° + tag\_angle\_world - yaw$. Here, the 90- degree adjustment aligns with the car's direction along the y-axis.

4.2 Using Trigonometry

We also investigated how to use trigonometry to estimate the car position. In general, this is a two-step procedure. The first step is to use θ_{AP}^{W} and θ_{AP}^{cam} to estimate car's orientation in world coordinate (θ_{car}^{W}). The second step is to use $(x_{AP}^{cam}, y_{AP}^{cam}, \theta_{AP}^{cam})$, $(x_{AP}^{W}, y_{AP}^{W}, \theta_{AP}^{W})$ and θ_{car}^{W} to estimate x_{car}^{W} and y_{car}^{W}. However, this approach is very complicated, we categorized more than 20 scenarios and discussed them separately. For details, you may refer to function `get_car_world_coordinate_trig` in `hw2_algorithm.py`.

5 Noise Handling

Our approach to reducing noise is straightforward yet effective. We observed that camera noise increases when the tag is detected near the edge of the screen or at greater distances from the camera (around 1.5 meters away). We suspect this noise is largely due to limitations in the camera calibration, particularly at the screen edges where distortion is more pronounced, affecting both position and distance accuracy. Our solution is to adjust the error threshold when these cases arise. For example, when the car is rotating, we increase the error threshold, making it easier for the car to meet the requirements. Similarly, if the next waypoint is far away, we raise the threshold, allowing the car to proceed more smoothly.

6 What to do when the car cannot detect any tags

Our primary goal is to assign the car a tag at every position it reaches. This way, the car can continuously determine its location by referencing the AprilTag. However, there are still edge cases where the car cannot detect any tags—for example, when it rotates to locate the next AprilTag at the final two waypoints. As explained in the PID controller section, we use the PID controller's output to estimate the car's movement. If no tag is detected, we apply this output to the current position to update the car's estimated position. While there may be some discrepancies between the ideal and actual positions due to hardware disturbances or other factors, this method has proven to be quite reliable.

7 Trajectory and Moving Distance

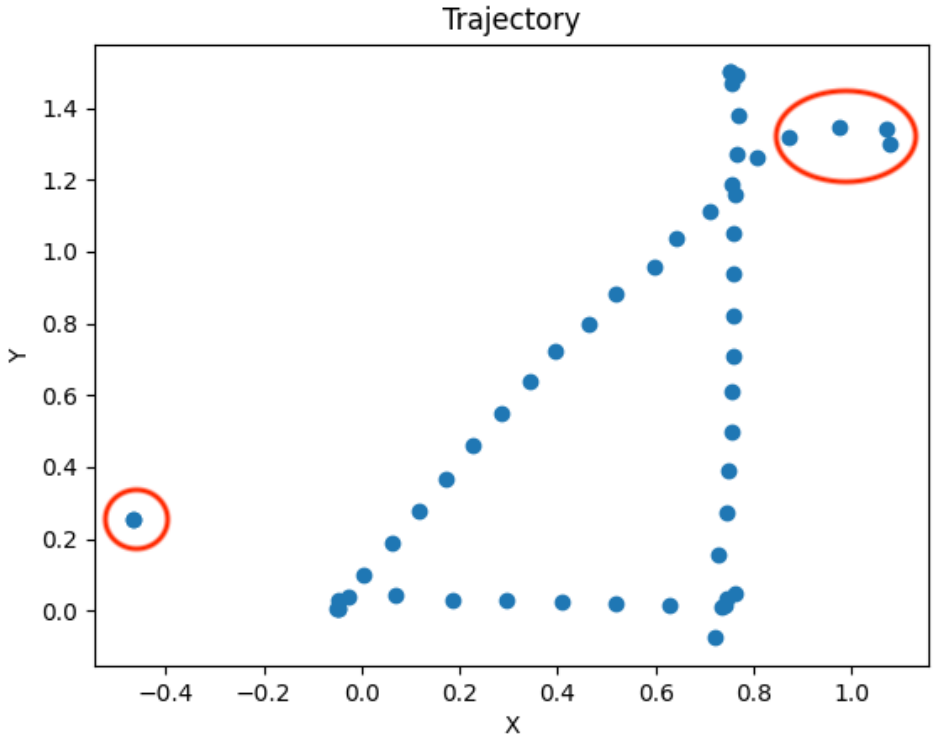

Figure 4: Trajectory

Our unit distance is 0.75m as shown in Figure 2. Figure 4 shows the trajectory. The dots red circles can be considered outliers. This is because the car was at (0.75, 1.5) and used tag at (-0.16, -0.165) to estimate the car position. The car estimation is not accurate because of the long distance between the camera and the tag.

If we exclude these outliers, the total moving distance is 4.39m(5.85 units), and total rotation is 364.64 °

8 Performance Evaluation

8.1 Smoothness of Car Movement

While the car is moving, the movement is pretty smooth. During the whole experiment, the car stops for 2 seconds after moving a certain distance, we designed this way because we want the camera have enough time to collect the latest April tag's information.

8.2 Location Error

From the video, the location error is very small(I would say less than 3 cm). The car can completely cover the marks that we set on the floor. Table 1 shows the location error according to car pose estimation.

waypoints	car pose estimation	error
(0.75, 0, 0°)	(0.73, 0.01, -5.22°)	(0.02, 0.01, -5.22°)
(0.75, 1.5, 180°)	(0.75, 1.47, -177.08°)	(0, 0.03, 2.92°)
(0, 0, 0°)	(-0.048, 0.030, 3.95°)	(0.048, 0.030, 3.95°)

Table 1: Location Error According to Car Pose Estimation – Using Meter and Degree

8.3 Reflection on Overall Performance

The overall performance of this experiment is pretty good. The location errors all fall into an acceptable range. However, we may use more April tags to reduce the number of outliers in car pose estimation.