Lab 3 – Inverse Kinematics

Patrick Nguyen, Wei Jiang, Nikko Gajowniczek

- 1.1)

$$\begin{bmatrix} \dot{x}_R \\ \dot{y}_R \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ \frac{r}{d} & -\frac{r}{d} \end{bmatrix} \begin{bmatrix} \dot{\phi}_l \\ \dot{\phi}_r \end{bmatrix} = \begin{bmatrix} \frac{\partial x_R}{\partial \dot{\phi}_l} & \frac{\partial x_R}{\partial \dot{\phi}_r} \\ \frac{\partial y_R}{\partial \dot{\phi}_l} & \frac{\partial y_R}{\partial \dot{\phi}_r} \\ \frac{\partial \theta}{\partial \dot{\phi}_l} & \frac{\partial \theta}{\partial \dot{\phi}_l} \end{bmatrix} \begin{bmatrix} \dot{\phi}_l \\ \dot{\phi}_r \end{bmatrix}$$

- 1.2) To create a trajectory, we need to create a set of points the robot must travel to, after manually recording those, we will have to figure out how to get the robot to move towards those waypoints, this can be difficult with imprecise odometry like that from lab 2. We plan on using precise GPS location to calculate the robot's precise error from the waypoint and minimize the error like that,

- 2.5) We fixed the issue of bearing error rapidly switching between positive and negative by implementing an epsilon error to count anything within +-0.01 as 0. This eliminated jittering.

- 3.2) First, bearing turn proportion is calculated based on how much error there is, and it is capped to 85% so the robot can also move forward for smooth driving. The remaining possible proportion for driving straight is 1-bearing proportion, so between 15 and 100%. This remaining value is multiplied by how close the distance is to epsilon, so when it reaches the waypoint, it stops moving forward. The final part is to implement the heading proportion which activates only when within 2 * distance epsilon. At the same time, this bearing proportion dampens the previously calculated bearing and distance proportions to prioritize only orienting the correct way once at the waypoint.

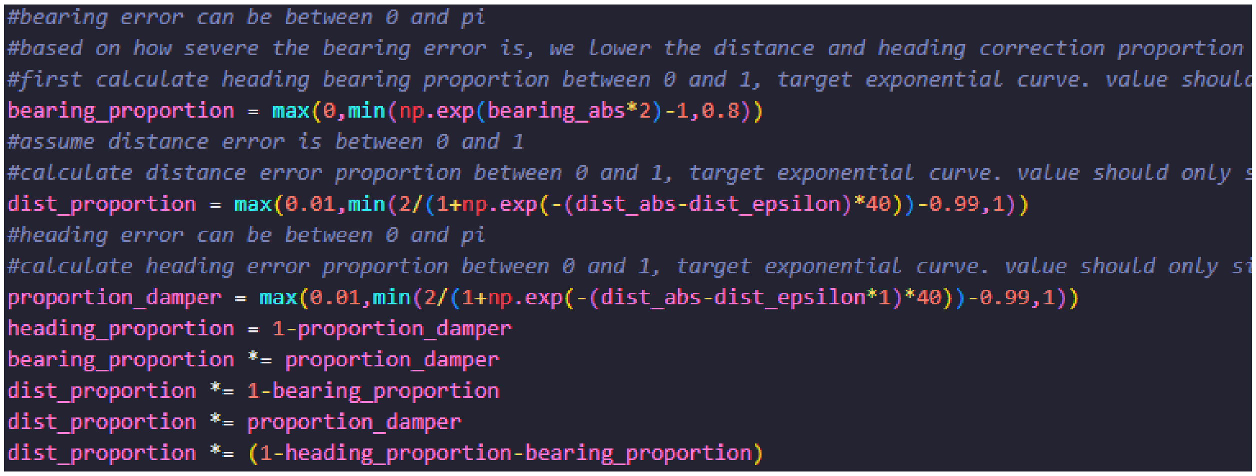
2. The role of the position error is that it calculates the distance from your current position and the goal point so that the robot can minimize the distance between itself and the waypoint.

3. The role of the heading error is that it is determining how off the robot is from the desired orientation once reaching the waypoint.

4. The role of the bearing error is to determine how far off in terms of orientation is the robot from being able to simply move forward and reach the waypoint. This is the first thing that is minimized before driving forward, otherwise the robot would move away from the waypoint.

5. An issue that we encountered for Part 3 was that we struggled to incorporate the heading error control with the proportional control to the others. With only two proportions, bearing and heading, it was easy to make the parts add up to equal 1. Once adding the heading error proportion, it became much more complicated to correctly scale each proportion given any range of values. Our odometry from Lab 2 seemed to consistently underestimate the amount turned by the robot, the straight distance traveled seemed to be very accurate, but even after the first turn our robot hit the wall due to improper angle calculation.

6. The equations for the feedback controller from the implementation in code are as follows:



```
#bearing error can be between 0 and pi
#based on how severe the bearing error is, we lower the distance and heading correction proportion
#first calculate heading bearing proportion between 0 and 1, target exponential curve. value shoul
bearing_proportion = max(0,min(np.exp(bearing_abs*2)-1,0.8))
#assume distance error is between 0 and 1
#calculate distance error proportion between 0 and 1, target exponential curve. value should only
dist_proportion = max(0.01,min(2/(1+np.exp(-(dist_abs-dist_epsilon)*40))-0.99,1))
#heading error can be between 0 and pi
#calculate heading error proportion between 0 and 1, target exponential curve. value should only
proportion_damper = max(0.01,min(2/(1+np.exp(-(dist_abs-dist_epsilon*1)*40))-0.99,1))
heading_proportion = 1-proportion_damper
bearing_proportion *= proportion_damper
dist_proportion *= 1-bearing_proportion
dist_proportion *= proportion_damper
dist_proportion *= (1-heading_proportion-bearing_proportion)
```

Knowing the domain of values for each proportion calculation, we fine tuned curves that produce responses similar to our desired qualitative effects.

7. Altering the constants would consist of basically scaling the domain to increase/decrease sensitivity to these different errors. It would cause the robot to primarily rotate more while also driving forward, creating more winding paths.

8. If the gain constants are increased too much, it will cause the system to become unstable, values will start to be clipped, and the robot will jitter.

9. We could add another proportion to this controller that forces the robot to turn right based on the brightness in front of it, and quickly drops to zero once facing away from the obstacle.

10. Around 5 hours were spent on coding and designing the algorithm, another 1-2 hours were spent fine tuning variables and testing performance.