دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)
دانشکده مهندسی برق

آز کنترل دیجیتال

# گزارشکار پروژه پایانی کنترل دیجیتال

اعضای گروه:
مهسا پنجه باشی
علی نظامی وند چگینی
محمدصادق رضوانی
علی رمضان زاده

استاد درس:
جناب آقای ستایشی

تیر 1403

# چکیده

در این پروژه، به پیاده‌سازی کنترلر PID برای هدایت و کنترل حرکت ربات e-puck در محیط شبیه‌سازی Webots پرداخته‌ایم. ربات باید از نقطه شروع تعیین‌شده در یک ماز آغاز به حرکت کرده و دو منطقه دنبال کردن خط را پیدا کند. پس از دنبال کردن خط، ربات به ماز بازگشته و به نقطه شروع خود بازمی‌گردد. الگوریتم DFS برای حل مسائل مسیریابی استفاده شده‌اند. ربات با استفاده از سنسورهای زمین و فاصله، محیط اطراف خود را شناسایی و برای حرکت دقیق، کنترلر PID را به‌کار می‌گیرد. در این پروژه، چالش‌هایی مانند وابستگی تسک‌ها و نیاز به کالیبراسیون دقیق مطرح شد که با استفاده از تکنیک‌های همگام‌سازی و تنظیم دقیق ضرایب کنترلر، بهبود یافتند. نتایج شبیه‌سازی نشان‌دهنده موفقیت‌آمیز بودن پیاده‌سازی‌ها و بهبود عملکرد ربات است. این پروژه نشان می‌دهد که ربات e-puck با الگوریتم‌های مناسب می‌تواند در محیط‌های پیچیده با دقت بالا عمل کند.

# فهرست مطالب

صفحه

# فصل اول

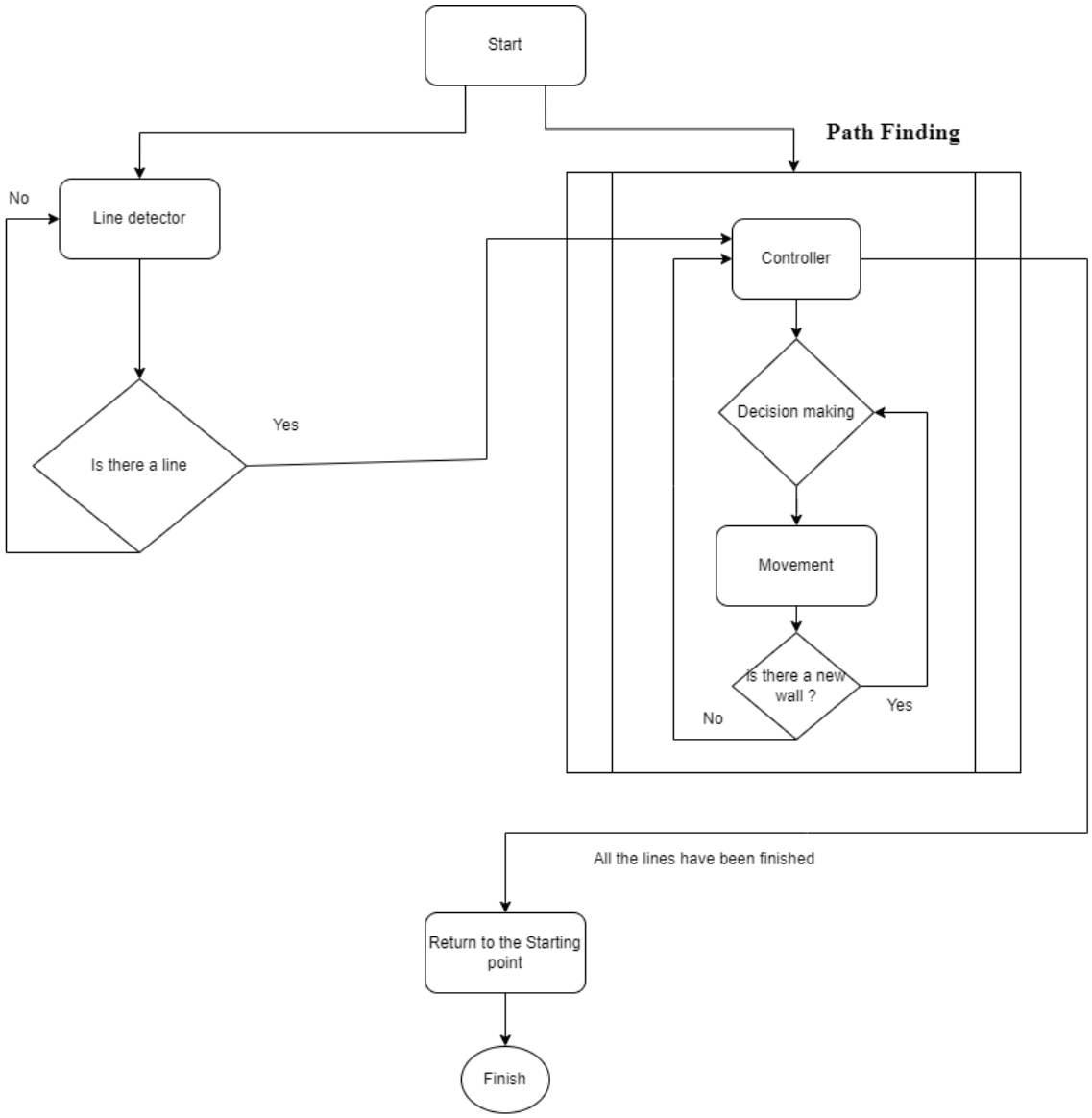
## مقدمه

# مقدمه

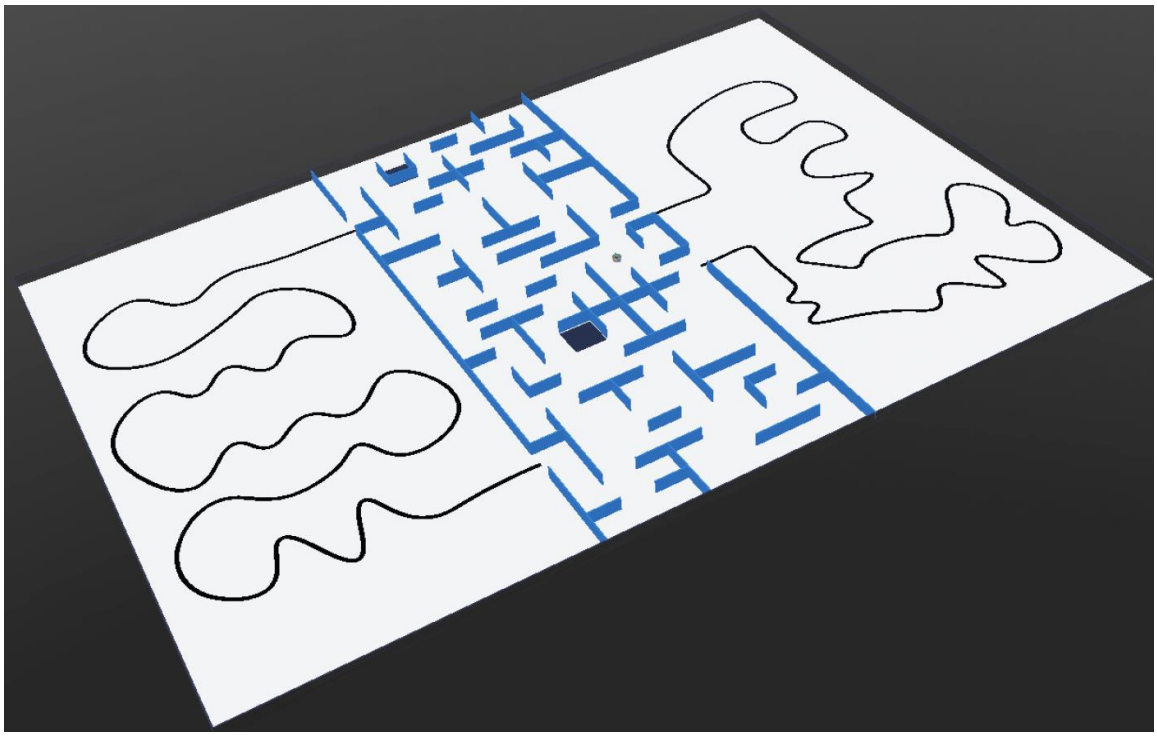
رباتیک به عنوان یکی از پیشروترین شاخه‌های علم و فناوری، نقش مهمی در توسعه سیستم‌های دیجیتال و هوشمند دارد. ربات‌ها امروزه در بسیاری از صنایع و کاربردهای مختلف، از جمله تولید، خدمات، پزشکی و حتی زندگی روزمره ما، نقش مهمی ایفا می‌کنند. یکی از ابزارهای رایج آموزشی و تحقیقاتی در این حوزه، ربات‌های کوچک و چندمنظوره e-puck هستند که به‌طور گسترده‌ای در دنیای آکادمیک و مسابقات رباتیک مورد استفاده قرار می‌گیرند.

مفاهیم آموخته شده در درس کنترل دیجیتال مانند طراحی کنترلر، نقش اساسی در بهبود عملکرد سیستم‌های رباتیک دارند. استفاده از کنترلرهای PID (تناسبی-انتگرالی-مشتقی) به دلیل سادگی در پیاده‌سازی و قابلیت تنظیم پارامترها، انتخاب مناسب و بهینه ای برای کنترل دقیق حرکت ربات‌ها به شمار می‌رود.

در این پروژه، به بررسی و پیاده‌سازی کنترلر PID برای هدایت و کنترل حرکت ربات e-puck در محیط شبیه‌سازی Webots، انجام وظایف داده شده و رفع مشکلات پیش‌آمده با روش های مختلف حل مسئله پرداخته‌ایم. این پروژه شامل دو تسک اصلی است: مسیریابی در ماز و دنبال کردن خط.




**شکل 1-1- فلوچارت و الگوریتم کلی برای مسیریابی و تعقیب خط ربات در کل مسیر داده شده**



**شکل 1-2- نقشه محیط شبیه‌سازی**

در ادامه، اهداف پروژه و پروسه رسیدن به آن را بررسی می‌کنیم.

# 1-1- اهداف پروژه

هدف کلی این پروژه، به این صورت است که ربات باید از نقطه شروع تعیین‌شده در یک ماز آغاز به حرکت کرده و شروع به مسیریابی کند تا دو منطقه دنبال کردن خط را پیدا کند. این مناطق شامل یک خط سیاه روی زمینه سفید هستند که ربات باید آن‌ها را دنبال کند. پس از اتمام وظیفه دنبال کردن خط، ربات باید به ماز برگشته و به نقطه شروع خود بازگردد.

وظایف اصلی پروژه عبارتند از:

- **مسیریابی ماز و دنبال کردن خط**

ربات باید با استفاده از الگوریتم‌های مناسب برای کشف ماز و یافتن مناطق دنبال کردن خط، از نقطه شروع تعیین‌شده در ماز آغاز به حرکت کند، پس از ورود به منطقه دنبال کردن خط باید خط سیاه روی زمینه سفید را دنبال کند، سپس به ماز بازگردد و پس از اتمام هر دو مسیر خط، به نقطه شروع بازگردد.

- **پیاده‌سازی کنترلر PID**

با استفاده از روش‌های گسسته‌سازی مناسب مانند اویلر یا توستین، کنترلر طراحی می‌شود. یک ساختار کلاس برای کنترلر PID ایجاد کرده و از آن در جهت وظایف مختلف مانند حرکت مستقیم، چرخش به چپ و راست و تضمین دقت در حرکت از آن استفاده می‌کنیم. درواقع این کنترلر به اطمینان از این که وظایف به‌طور کارآمد و دقیق انجام شوند، کمک می‌کند. (مدل‌سازی وظیفه دنبال کردن خط به عنوان یک مسئله کنترلی)

## 1-2- شناخت ربات مورداستفاده

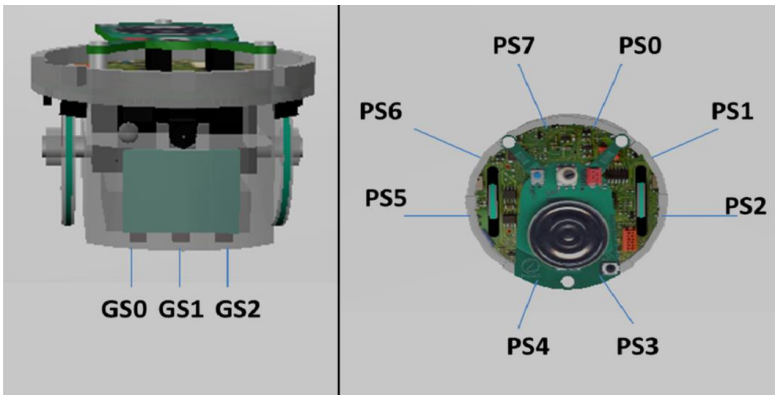
ربات e-puck، ابزار مناسبی برای پیاده‌سازی و تست الگوریتم‌های پیچیده کنترل و مسیریابی در محیط‌های مختلف است. این ربات در این پروژه برای انجام وظایف حل ماز و دنبال کردن خط مورد استفاده قرار گرفته و نتایج عملی به دست آمده از آن، نشان‌دهنده قابلیت‌های بالای این پلتفرم در آموزش و تحقیقات رباتیک است. از ویژگی‌های اصلی ربات e-puck که نکته کلیدی در کد نویسی برای آن است، سنسور های آن میباشد.

در راستای کدنویسی برای ربات e-puck، لازم است مشخصات مکانیکی آن را بشناسیم.

1. سنسورها

- سنسور زمین: با قابلیت تشخیص رنگ از طریق IR، خطوط روی زمین را تشخیص می دهد و به ربات کمک می کند تا خطوط را دنبال کند یا تقاطع ها را شناسایی کند.
- سنسور مجاورتی: فاصله از اشیا را اندازه گیری میکنند و موانع اطراف ربات را تشخیص می دهند.
- انکودر: چرخش چرخ ها را اندازه گیری می کنند که می توان از آن برای کنترل دقیق چرخش و محاسبه مسافت طی شده استفاده کرد.

2. عملگرها: کنترل حرکت چرخ‌ها برای حرکت به جلو، عقب، چرخش به چپ و راست.




**شکل 1-3- ربات e-puck و سنسور های آن**

# فصل دوم

# گام های پیاده‌سازی

# گام های پیاده‌سازی

## 2-1-الگوریتم DFS

برای ساده سازی پیمایش ماز، مسیر و دنیای داده شده را به یک درخت تبدیل می کنیم. این درخت با شاخه هایش، یا این والد با فرزندانش، نمایشی ساده شده از مسیر پیچیده ما هستند. دو الگوریتم معرفی شده برای حل این ماز BFS و DFS هستند. الگوریتم مورداستفاده در این پروژه DFS می باشد که مخفف عبارت Depth-First Search است، یعنی بر روی عمیق شدن در مسیر تمرکز می کند و اولویت آن **عمق** است.

یک کلاس Tree Node تشکیل می‌دهیم که حرکت ربات را ذخیره می‌کند و دارای حافظه برای تشخیص بلوک تکراری می‌باشد.

بدین ترتیب ربات با الگوریتم DFS، شروع به حرکت و گذر از موانع می‌کند. همزمان با این پیمایش، اگر ربات خط را تشخیص دهد، اولویت ها تغییر می‌کنند. ربات با تشخیص اولین خط، ماز را پیمایش می‌کند تا به سر دیگر خط، متناظر با خط اولیه برسد. سپس آن سر خط را دنبال می‌کند، تا به انتهای آن و بلوکی که برای اولین بار در آن خط را تشخیص داد، برسد. سپس ربات به پیمایش ماز و گذر از موانع ادامه می‌دهد تا مجددا مسیر خط دوم را پیدا کند و آن را بپیماید.

# 2-2-طراحی کنترلر PID

کنترل کننده PID پیوسته در حوزه s به صورت زیر است:

$$C(s) = K_p + \frac{K_i}{s} + K_d s$$

## 2-2-1- گسسته سازی

**گسسته‌سازی کنترل‌کننده PID با استفاده از روش اویلر:**

- برای ترم انتگرالی، انتگرال را با یک جمع تقریب می‌زنیم:

$$\int e(t)\,dt \approx \sum e[k] \cdot T$$

که در آن T دوره نمونه‌برداری است.

- برای ترم مشتق‌گیری، مشتق را با یک تفاضل تقریب می‌زنیم:

$$\frac{de(t)}{dt} \approx \frac{e[k] - e[k-1]}{T}$$

**کنترل‌کننده PID گسسته:**

با استفاده از این تقریب‌ها، کنترل‌کننده PID گسسته به صورت زیر خواهد بود:

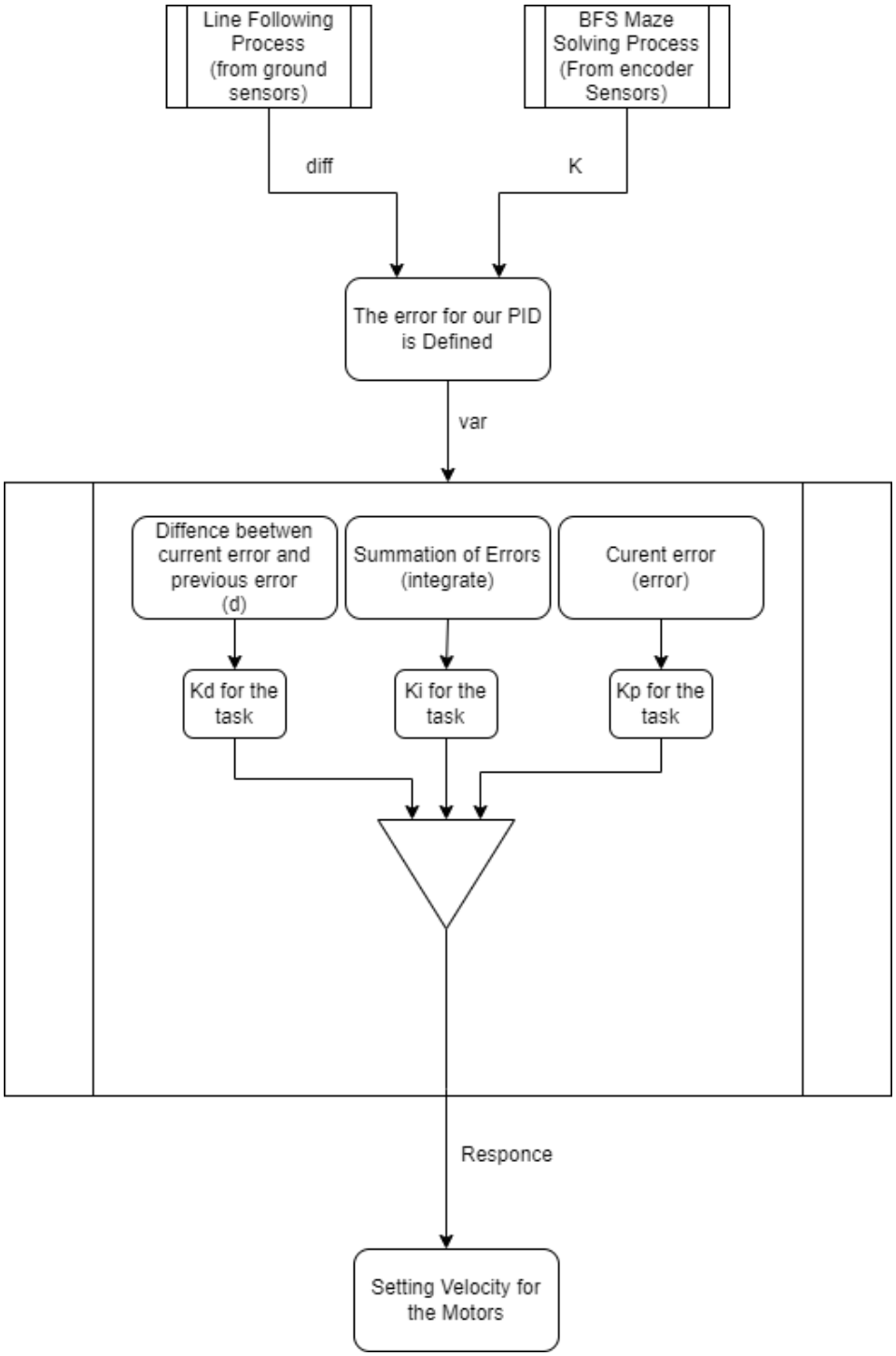
$$u[k] = K_p e[k] + K_i \sum e[k] \cdot T + K_d \frac{e[k] - e[k-1]}{T}$$

با انتخاب T=1، فرمول نهایی به این صورت خواهد بود:

$$u[k] = K_p e[k] + K_i \sum e[k] + K_d (e[k] - e[k-1])$$

باید محاسبات PID را تبدیل به مسئله کنترل کنیم و مسئله را بر مبنای ارور حل کنیم. در واقع بر مبنای ارور، PID را تشکیل می‌دهیم!

در ادامه فلوچارت مربوط به الگوریتم طراحی PID را تشکیل می‌دهیم.




**شکل 2-1- فلوچارت طراحی کنترلر PID و پیاده سازی آن بر روی سیستم**

### 2-2-2- کلاس PID_Controller

با __init__def مقادیر اولیه کنترلر PID را تنظیم می کنیم.

$K_P$ ضریب تناسبی ، $K_D$ ضریب مشتق گیر و $K_I$ ضریب انتگرال گیر ما می‌باشد و با توجه به نحوه ی عملکرد ربات در شبیه‌سازی و tune مناسب، مقادیر این ضرایب را تنظیم می کنیم.

**خطا در این محاسبات یک مفهوم مهم دارد: اختلاف موقعیت فعلی ربات با موقعیت دلخواه**

اکنون بخش اصلی کد کنترلر PID را به طور خلاصه شرح می‌دهیم و در فصل بعدی با جزئیات بیشتر به توضیح کد های شبیه‌سازی می‌پردازیم.

در بدنه‌ی کد، Error خطای محاسبه شده در سیکل فعلی، و pre_error خطای محاسبه شده در سیکل قبلی اجرای شبیه‌سازی می‌باشد. Integerate پارامتری است که مجموعه خطا های ما در آن ذخیره شده و center، مبنا و حالت مطلوبی است که می خواهیم به آن برسیم.

```
  def calculate(self, val):
      self.error = self.center - val
      self.integrate += self.error
      d = self.error - self.pre_error

      res = (self.kp * self.error ) + (self.ki*self.integrate) + (self.kd * d )
      self.pre_error = self.error

      return res
pid = PID_Controller()
```

error یا همان خطا را به صورت اختلاف مبنای فعلی (سنسور وسط ربات) و مقدار اندازه‌گیری شده تعریف می‌کنیم، بدیهی است که این مقدار باید جبران شود.

### 2-2-3- ضرایب PID

برای سه نوع حرکت به طور جداگانه ضرایب کنترلر PID را بدست می‌آوریم:

حرکت رو به جلو و عقب، حرکت چرخشی، و دنبال کردن خط.

حدود این ضرایب را می‌توان با tune کردن از طریق سیمولینک و راه های محاسباتی دیگر بدست آورد، اما می‌دانیم این مقادیر با مقادیر عملی متفاوت هستند و در هر صورت برای دقت و صحت بیشتر باید با تست شبیه‌سازی به روش عملی این ضرایب را محاسبه کنیم.

- **ضریب $k_p$** بیانگر ضریب تناسبی کنترلر PID می‌باشد، و افزایش آن سبب بالا رفتن سرعت سیستم می‌شود که افزایش بالازدگی را به همراه دارد که از معایب آن می‌باشد. این اورشوت در حدی می‌تواند ایجاد مشکل کند که ربات هنگام تعقیب خط ممکن است از خط خارج شود و مسیر خود را گم کند.
- **ضریب $k_d$** بیانگر ضریب مشتق‌گیر کنترلر PID می‌باشد، و به منظور بهبود پاسخ گذرا و حالت اولیه سیستم به کار می‌رود. افزایش آن سبب کم شدن نوسانات و جلوگیری از حرکت زیگزاگی می‌شود، اما در عمل سرعت ربات را کند می‌کند.
- **ضریب $k_i$** بیانگر ضریب انتگرال‌گیر کنترلر PID می‌باشد، افزایش آن به بهبود خطای حالت دائم کمک می‌کند، اما سبب کند شدن سیستم و حتی ایجاد ناپایداری می‌شود. این پدیده در حرکت چرخشی ربات ایجاد مشکل می‌کند و می‌تواند باعث شود ربات در چرخش از مسیر خود منحرف شود.

مقادیر عددی ضرایب PID که با تست شبیه‌سازی و به روش عملی به دست آمده اند:

- Line Following:

  $K_p = 1.15$

  $K_d = 0.15$

  $K_i = 0.001$

- Movement (Forward & Backward)

  $K_p = 2$

  $K_d = 0.2$

  $K_i = 0.01$

- Rotation (90° , clockwise & counter clockwise)

  $K_p = 1.5$

  $K_d = 0.3$

  $K_i = 0.04$

## 2-3-تشخیص دیوار

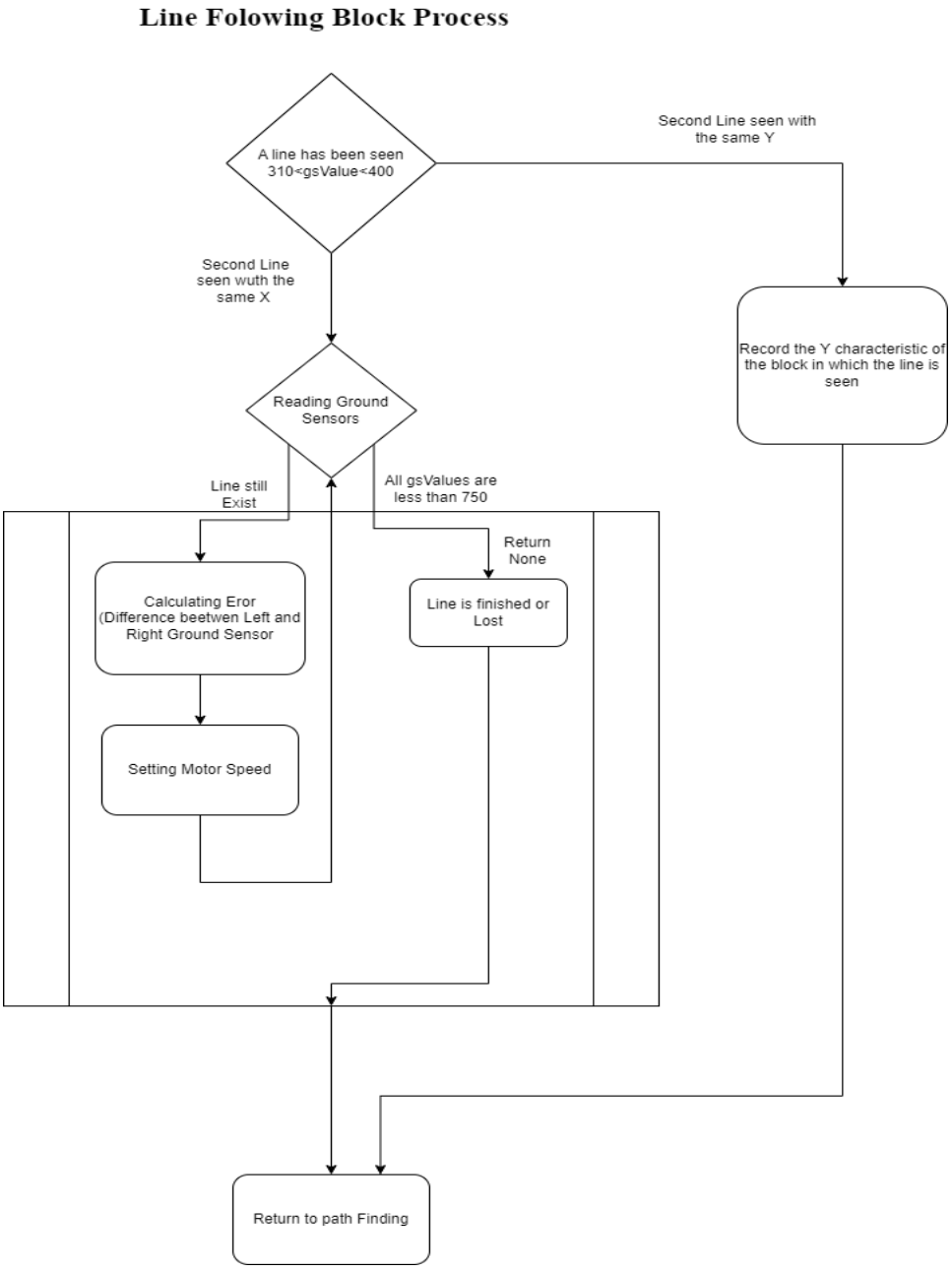
تشخیص دیوار با استفاده از سنسور های ps که باید مقداری معین داشته باشند، انجام می‌گیرد. ربات بر مبنای آن، حرکت به جهت دیگری که در اولویت بعدی قرار دارد را اجرا می‌کند و مسیر خود را پیدا می‌کند. در این‌جا از مفهوم جهت مرجع استفاده کرده‌ایم؛ جهتی که ربات در آن روشن می‌شود و شروع به حرکت می‌کند، سمت راست آن، همیشه شمال است. در ابتدا 90 درجه ساعت‌گرد می‌چرخد و جهت آن تثبیت می‌گردد.

## 2-4-تشخیص خط

برای تشخیص خط، از سنسور های زمین (سنسور های کف) استفاده میکنیم و مقادیر آن را با تابعی که gs value را می‌خواند، می‌گیریم و عدد آن را مقایسه می‌کنیم با عدد متناظر با رنگ مورد نظر. به عنوان مثال حدود عدد 310 به بالا برای رنگ مشکی، و حدود عدد 760 برای رنگ سفید می‌باشد.

## 2-5- تعقیب خط

میان سه سنسور زمین، ست پوینت ما سنسور وسط است. با توجه به این‌که کدام یک از سنسور های جانبی و کناره فعال می‌شوند و خط را می‌بینند، متوجه انحراف ربات می‌شویم و بر مبنای آن با اعمال سرعت مناسب به موتورها، ربات را در مسیر اصلی حفظ می‌نماییم.




**شکل 2-2- فلوچارت مربوط به الگوریتم دنبال کردن خط**

# فصل سوم

# توضیح کد

# توضیح کد

در ابتدا نیاز به import برخی از کلاس‌ها و ماژول‌ها داریم.

- کلاس Robot به ما این امکان را می‌دهد تا داده‌های سنسورها را بخوانیم، عملگرها (موتورها) را کنترل کنیم و با دیگر اجزای ربات تعامل داشته باشیم.
- ماژول threading بخشی از کتابخانه استاندارد پایتون است و به ما امکان می‌دهد تا رشته‌ها (واحدهای کوچکتر یک فرآیند) را در یک برنامه پایتون ایجاد و مدیریت کنیم. به عنوان مثال، یک‌بخش داده‌ها را از سنسورها بخواند وبخش دیگر فرآیند حرکت ربات را انجام دهد.
- ماژول numpy نیز برای محاسبات مربوط به مسافت طی شده مورد استفاده قرار گرفته است.
- ماژول os نیز برای خاتمه دادن به برنامه استفاده شده‌اند.

- کلس TreeNode: یک کلاس برای تعریف دیاگرام درختی که در ادامه لازم داریم با استفاده از روابط بین این دیاگرام نقشه مسیرها را تعیین کنیم.
  - متد add_child: این متد یک گره فرزند ( زیر شاخه ) را به گره فعلی اضافه می‌کند و مرجع والد را برای گره فرزند تنظیم می‌کند.
  - متد show_tree: این متد برای چاپ ساختار درخت به صورت یک نمایش سلسله مراتبی استفاده می‌شود. این روش برای نمایش گرافیکی درخت به کار می‌رود، به طوری که هر سطح درخت با فاصله‌های بیشتری نسبت به سطح بالاتر نمایش داده می‌شود.
  - متد get_level: این متد سطح گره را در درخت محاسبه می‌کند. سطح هر گره تعداد والدین بین آن گره و ریشه درخت است. برای این منظور، متد با پیمایش از گره فعلی تا ریشه، تعداد والدین را می‌شمارد.

- کلس PID_Controller:

این کلاس یک کنترل‌کننده PID را پیاده‌سازی می‌کند که برای تنظیم و کنترل سیستم‌ها به کار می‌رود. پارامترهای kp، ki و kd ضرایب کنترلی هستند و متغیرهایی مانند error، pre_error و integrate برای مدیریت خطاها و محاسبات کنترلی استفاده می‌شوند.

- تابع کالیبراسیون calibr: این تابع به نام calibr، خطای اندازه‌گیری شده توسط سنسورها را کالیبره می‌کند و جهت حرکت را بر اساس ورودی direction تنظیم می‌کند. با توجه به جهت حرکت (f برای جلو و b برای عقب)، سرعت موتورها تنظیم می‌شود تا خطا به حد مطلوب برسد.
- تابع read_encoder: این تابع به نام read_encoder، مقادیر انکودرها را می‌خواند. با استفاده از یک حلقه، مقادیر دو انکودر را در لیستی جمع‌آوری کرده و در نهایت آن را برمی‌گرداند.
- تابع move_func: این تابع به نام move_func برای حرکت و چرخش ربات استفاده می‌شود. پارامترهای block_rate برای میزان حرکت، rotation برای زاویه چرخش و direction برای جهت چرخش (ساعت‌گرد یا پادساعت‌گرد) استفاده می‌شوند.
- ابتدا، مقادیر انکودرها خوانده می‌شوند. در صورت نیاز به چرخش، زاویه به رادیان تبدیل شده و سپس ربات با استفاده از کنترل‌کننده PID چرخش را انجام می‌دهد.
- در مرحله بعد، حرکت به سمت جلو یا عقب با استفاده از کنترل‌کننده PID انجام می‌شود تا میزان حرکت مورد نظر طی شود. در هر مرحله، مقادیر انکودر به‌روز شده و سرعت موتورها تنظیم می‌شود.
- تابع get_gs_values: این تابع به نام get_gs_values ، مقادیر سنسورهای زمین (gs) را می‌خواند. با استفاده از یک حلقه، مقادیر سه سنسور را در یک لیست جمع‌آوری کرده و در نهایت آن را برمی‌گرداند.

- تابع line_follower: این تابع به نام line_follower، برای دنبال کردن خط طراحی شده است. سه سنسور زمین فعال شده و سرعت موتورها به صفر تنظیم می‌شود، سپس یک تاخیر کوتاه برای راه‌اندازی صحیح اعمال می‌شود.
- تابع get_values: این تابع به نام get_values، مقادیر سنسورهای زمین را می‌خواند. اگر مقادیر هر سه سنسور بیش از 750 باشد، یعنی خط پایان یافته یا گم شده است و None برمی‌گرداند. در غیر این صورت، تفاوت مقادیر سنسور اول و سوم را تقسیم بر 200 کرده و برمی‌گرداند.
- تابع follower: این تابع به نام follower، خط را دنبال می‌کند. مقدار خطا (diff) را از سنسورها می‌گیرد. اگر خطا None باشد، line_flag را False کرده و حلقه را متوقف می‌کند. در غیر این صورت، میزان مورد نیاز را محاسبه و سرعت موتورها را تنظیم می‌کند.
- تابع line_detector: این تابع خط را تشخیص می‌دهد و در صورت تشخیص، پرچم مربوطه را روشن می‌کند. همچنین از دو پرچم برای ارسال و دریافت سیگنال به کنترلر استفاده می‌شود.
- تابع make_node: این تابع وظیفه دارد تا گره‌ها را به دیاگرایم درختی که از قبل طراحی کرده‌ایم در جای مشخص بیافزاید.
- تابع driver_controller: این ماژول را طراحی کردیم تا با گرفتن جهت حرکتی مورد نیاز، دستورات مورد نیاز را برای حرکت در جهت و راستای مورد نظر اعمال کند.
- تابع Movement_controller: این ماژول وظیفه دارد تا فرمان را از ماژول controller بگیرد و با توجه به فرمان کنترلر عملیات های مورد نیاز جهت پیش بردن الگوریتم ربات را انجام دهد.
- تابع path_finder: در برخی موارد که ما نیازمندیم به خانه یا بلوک والد برگردیم، برای اینکه متوجه بشویم این بلوک در کدام سمت ربات قرار دارد از این ماژول استفاده می‌نماییم.
- متد read_sensors: این تابع مقادیر سنسورهای فاصله‌ی ما را می‌خواند.

- ماژول controller: این ماژول که ماژول اصلی ربات نیز می باشد وظیفه دارد تا در هر وهله عملیاتی با استفاده از اطلاعات ورودی شرایط ربات را بسنجد و دستور مناسب را به ماژول movement_controller ارسال نماید. در این ماژول سعی شده تا با در نظر داشتن اطلاعاتی که از قبل کسب کردیم و همچنین اهدافی که داریم بتوانیم بهینه‌ترین روش در جست و جوی نقشه را پیاده‌سازی کنیم.

به صورت هم زمان در این ماژول با خواندن فلگ line_flag، دستورات مربوطه به این پخش اجرا می‌شود که اعم است از

- ✓ تشخیص این که آیا در خط جدید قرار داریم یا خیر
- ✓ اگر در خطی قرار داریم که ورودی آن را قبلا مشاهده کرده‌ایم آن خط را پیدا کنیم

- همچنین در این ماژول در نظر داشتیم تا بتوانیم بعد از خارج شدن از فرآیند تعقیب خط، ربات را در مرکز بلوک کالیبره کنیم که این روش را با استفاده از دیوارهای اطراف آن بلوک پیاده‌سازی کرده‌ایم و در صورتی که محور Y ما که بعد از خارج شدن از خط ممکن است به صورت کامل از حالت کالیبر در بیاید، کالیبر نشود، از خانه‌ی والد به منظور کالیبر کردن محور عمودی کمک می‌گیریم و در ادامه فرآیند جست و جوی نقشه را ادامه می‌دهیم تا زمانی که همه‌ی خط‌ها را بپیماییم، این نکته حائز اهمیت می‌باشد که در تمامی عملیات‌های این ماژول اولویت‌های بررسی به صورت دینامیکی طوری تنظیم شده تا بهینه‌ترین جست و جو را برای یافتن خطوط داشته باشیم و همچنین بتوانیم استراتژی اصلی خود که وارد نشدن به سر اول خط می‌باشد را پیاده‌سازی کنیم، در تمامی عملیات‌ها در صورت نیاز کالیبراسیون ربات انجام گیرد.

- تابع back_to_home: زمانی که تمامی خط‌های مورد نیاز را طی کردیم، این متد اجرا می‌شود تا ربات از طریق نزدیک‌ترین مسیری که طی کرده بود، به نقطه‌ی شروع باز گردد.

- تابع show_tree: در نهایت، این تابع برای نمایش ساختار درختی نقشه استفاده می‌شود.

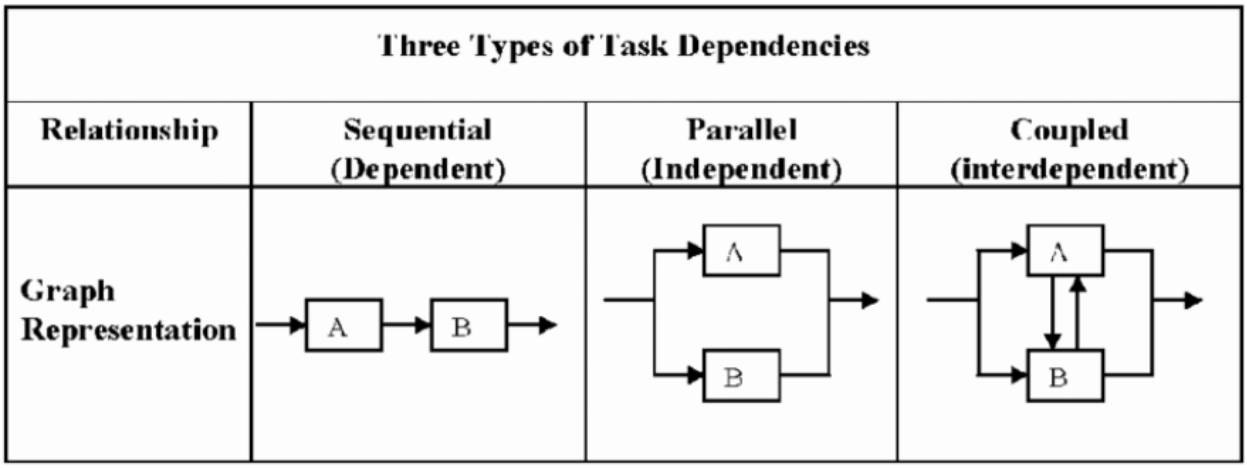
# فصل چهارم

# چالش های مطرح شده و راه حل آن

# چالش های مطرح شده و راه حل آن

## Task Dependency -1-4

برای رفع مشکل وابستگی تسک‌ها در این کد، به خصوص در حالتی که تسک‌ها به هم وابسته هستند ((coupled، از ترکیبی از threading و کنترل PID استفاده کردیم. در اینجا نحوه مدیریت وابستگی تسک‌ها و اطمینان از عملکرد صحیح ربات توضیح داده شده است.




**شکل 4-1- سه نوع وابستگی تسک ها نسبت به یکدیگر**

انواع وابستگی تسک‌ها و مدیریت آن:

1. تسک‌های ترتیبی (dependent)

این تسک‌ها وابسته به تکمیل تسک‌های قبلی هستند. در کد، این کار با ترتیب‌دهی صریح در داخل توابع و استفاده از تأخیرها برای اطمینان از تکمیل یک تسک قبل از شروع تسک بعدی مدیریت می‌شود.

- ✓ تابع move_func ربات را حرکت می‌دهد و منتظر می‌ماند تا حرکت کامل شود قبل از اینکه اجازه دهد تسک بعدی شروع شود.
- ✓ تابع delay برای ایجاد تأخیر زمانی بر اساس زمان‌بندی ربات استفاده می‌شود تا اطمینان حاصل شود که تسک‌ها به ترتیب تکمیل می‌شوند.

2. تسک‌های موازی (independent)

این تسک‌ها می‌توانند به صورت موازی اجرا شوند بدون اینکه به یکدیگر وابسته باشند. در کد، از threading برای مدیریت چنین تسک‌هایی استفاده می‌شود.

- ✓ line_detector و controller به صورت موازی در thread های جداگانه اجرا می‌شوند.

3. تسک‌های وابسته و متصل (Coupled)

این تسک‌ها به یکدیگر وابسته هستند و ممکن است نیاز به هماهنگی داشته باشند. مسیریابی و خواندن سنسورها توسط ربات به یکدیگر وابسته هستند.

- ✓ داده‌های مشترک و همگام‌سازی: متغیرهای مشترک مثل line_flag و line_flag_write برای هماهنگی بین بخش‌های مختلف کد استفاده می‌شوند. متغیر line_flag نشان می‌دهد که آیا ربات خطی را شناسایی کرده است یا خیر که بر تصمیمات حرکتی آن تأثیر می‌گذارد.

# Synchronizing -2-4

همان‌گونه که در بخش قبل توضیح دادیم، با توجه به تعداد بالای توابع مختلف و الزام به همزمان پیش بردن تسک ها و عملیات مختلف ربات، از وهله عملیاتی thread برای موازی جلو بردن تسک های کنترلر و تشخیص خط استفاده می‌کنیم.

همگام‌سازی (Synchronizing) به فرآیند هماهنگ‌سازی اجرای چندین تسک به منظور جلوگیری از مشکلاتی مانند شرایط مسابقه (race conditions) و حفظ یکپارچگی داده‌ها اشاره دارد. همگام‌سازی تضمین می‌کند که منابع مشترک به درستی مدیریت می‌شوند و تسک‌ها بدون تداخل با یکدیگر اجرا می‌شوند. در این کد از thread برای اجرای موازی توابع controller و line_detector استفاده شده است. این تردها به طور همزمان اجرا می‌شوند اما برای همگام‌سازی از join برای منتظر ماندن تا تکمیل هر ترد استفاده شده است.

- ✓ متغیرهایی مانند line_flag و line_flag_write برای هماهنگی بین تسک‌های مختلف استفاده می‌شوند. این متغیرها وضعیت‌های مختلف را ذخیره می‌کنند که تسک‌های مختلف می‌توانند بر اساس آن‌ها تصمیم‌گیری کنند.
- ✓ در مواردی که نیاز به محافظت از منابع مشترک و جلوگیری از تداخل همزمان وجود دارد، از قفل‌ها استفاده می‌شود. در این کد، می‌توان از threading.Lock برای ایجاد قفل و همگام‌سازی دسترسی به متغیرهای مشترک استفاده کرد (هرچند در کد فعلی قفل‌ها به طور صریح استفاده نشده‌اند، اما این یک روش رایج برای همگام‌سازی است).

```
lock = threading.Lock()


with lock:
#     critical section
    line_flag = True
```

## Calibration -3-4

ربات در حرکت خود، چه به صورت چرخشی چه حرکت مستقیم به سوی جلو یا عقب، نیاز به دقت بالا دارد که از مسیر اصلی خود خارج نشود و الگوریتم به خوبی اجرا شود. این دقت در جابجایی تا حد خوبی به وسیله کنترلر PID تضمین شده است؛ هرچند برای بهینه ترین شبیه‌سازی همچنان نیاز به کالیبراسیون داریم. کالیبراسیون با استفاده از دیوار های اطراف بلوک انجام می‌گیرد. ربات اول به دنبال دیوار میگردد تا خود را نسبت به آن کالیبره کند و اطمینان حاصل کند که در مرکز بلوک فرضی قرار دارد. فرایند مربوط به کالیبراسیون به وسیله تابع calibr انجام می‌شود که در فصل توضیح کد آن را به طور کامل شرح دادیم.

# فصل پنجم

# نتایج شبیه‌سازی و ارائه پیشنهادات

# نتایج شبیه‌سازی و ارائه پیشنهادات

ربات و کد کنترلر طراحی‌شده را در محیط شبیه‌سازی webots پیاده سازی کردیم و نتیجه مطلوب را از آن گرفتیم. ویدیوی این شبیه‌سازی در فایل پروژه پیوست شده‌است.

به طور خلاصه، نکات مهمی را مرور می‌کنیم که با استفاده از آن، قادر شدیم پروژه را بهینه‌تر کنیم:

- همانطور که در شبیه سازی مشاهده کردیم ربات توانست به درستی هر دو خط را شناسایی کند و به خانه اول باز گردد و در انتها نمودار درختی مربوط به مسیر های طی شده به هر بلوک و همچنین مختصات ورودی های هر خط را در خروجی چاپ کند.
- با تست های عملی متنوع و تغییر اولویت های گردش به صورت دینامیک توانستیم زمان اجرا را تا 60 درصد کاهش دهیم که بهینه سازی بسیار مناسبی برای این ربات با این مجموعه تسک ها میباشد.
- با ایجاد دو روش کالیبراسیون توانستیم خطای مکانی را تا 95 درصد و خطای زاویه ای را تا 99 درصد در هر بلوک کاهش دهیم و همچنین با ایجاد سیستم کالیبراسیون بعد خطوط توانستیم خطای زاویه ای را تا 99 درصد و خطای مکانی را تا 93 درصد کاهش دهیم.
- با موازی سازی دو تابع line_detector و controller کار تشخیص خط را بسیار ساده تر کردیم به این دلیل که دیگر لزومی به ایجاد توقف های زمانی و در نظر گرفتن این توقف ها در مراحل مختلف فرایند های تصمیم گیری ربات نبوده است.
- این نکته حائز اهمیت می باشد که این ربات می تواند با قرار گرفتن در هر نقطه ای از نقشه به شرطی که در مرکز بلوک خود باشد و با هر تغییر زاویه ای به شرطی که اختلاف آن با زاویه فعلی مضربی از 180 درجه باشد نیازمندی ها را ارضا کند.

- در برخی شرایط خاص با توجه به نحوه قرار گیری ربات ممکن است بهینه سازی که در مورد دوم به آن اشاره کردیم به ثمر نرسد، اما در اکثر حالات این تغییر اولویت جست و جو به صورت داینامیک باعث بهبود زمان عملکرد ربات می شود.
- با رصد کردن عدد خوانده شده از انکدر های موتور در هنگام چرخش، متوجه این موضوع شدیم که به هنگام حرکاتی که در آن یک موتور در جهت عکس دیگری حرکت می‌کند، با یک تاخیر در شروع به حرکت کردن یکی از موتور ها مواجه هستیم که همین دلیل سبب ایجاد خطا در بلند مدت می‌گردد.

## پیشنهادات:

- ✓ استفاده از GPS برای تسهیل و تسریع سیستم مسیر یابی و حرکنی ربات در ماز.
- ✓ استفاده از ماشین لرنینگ برای بهبود الگوریتم یافتن ورودی های مختلف خط ، میتوانیم از روش reinforcements learning نیز در این بخش استفاده کنیم.
- ✓ استفاده از چند PID مختلف برای رنج های مختلف حرکتی و حرکت های مختلف تا بتوانیم حرکت ربات را تسریع بخشیم و خطا را به حداقل برسانیم
- ✓ برای بهبود عملکرد ربات بطوری که در هر نقطه ای و در هر پوزیشنی بتواند به بهترین شکل نقشه را بپیماید میتوانم چهار تسک الگورتیم حرکت کردن، تشخیص خط،خواندن انکدر موتور،و تشخیص دیوار را موازی سازی کنیم، با توجه به کمبود وقت از انجام این امور در این پروژه صرف نظر شده است.

GitHub Repository:

https://github.com/AliNzmv/Maze-line-explorer-.git