



دانشگاه صنعتی امیرکبیر  
(پلی تکنیک 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 به کار 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.

1	فصل اول مقدمه.....
5	1-1- اهداف پروژه.....
6	2-1- شناخت ربات مورد استفاده.....
7	فصل دوم گام های پیاده سازی.....
8	1-2- الگوریتم DFS.....
9	2-2- طراحی کنترلر PID.....
9	1-2-2- گسسته سازی.....
12	2-2-2- کلاس PID_Controller.....
13	3-2-2- ضرایب PID.....
15	3-2- تشخیص دیوار.....
16	4-2- تشخیص خط.....
17	5-2- تعقیب خط.....
18	فصل سوم توضیح کد.....
23	فصل چهارم چالش های مطرح شده و راه حل آن.....
24	4-1- Task Dependency.....
26	2-4- Synchronizing.....
27	3-4- Calibration.....
28	فصل پنجم نتایج شبیه سازی و ارائه پیشنهادات.....

## فصل اول

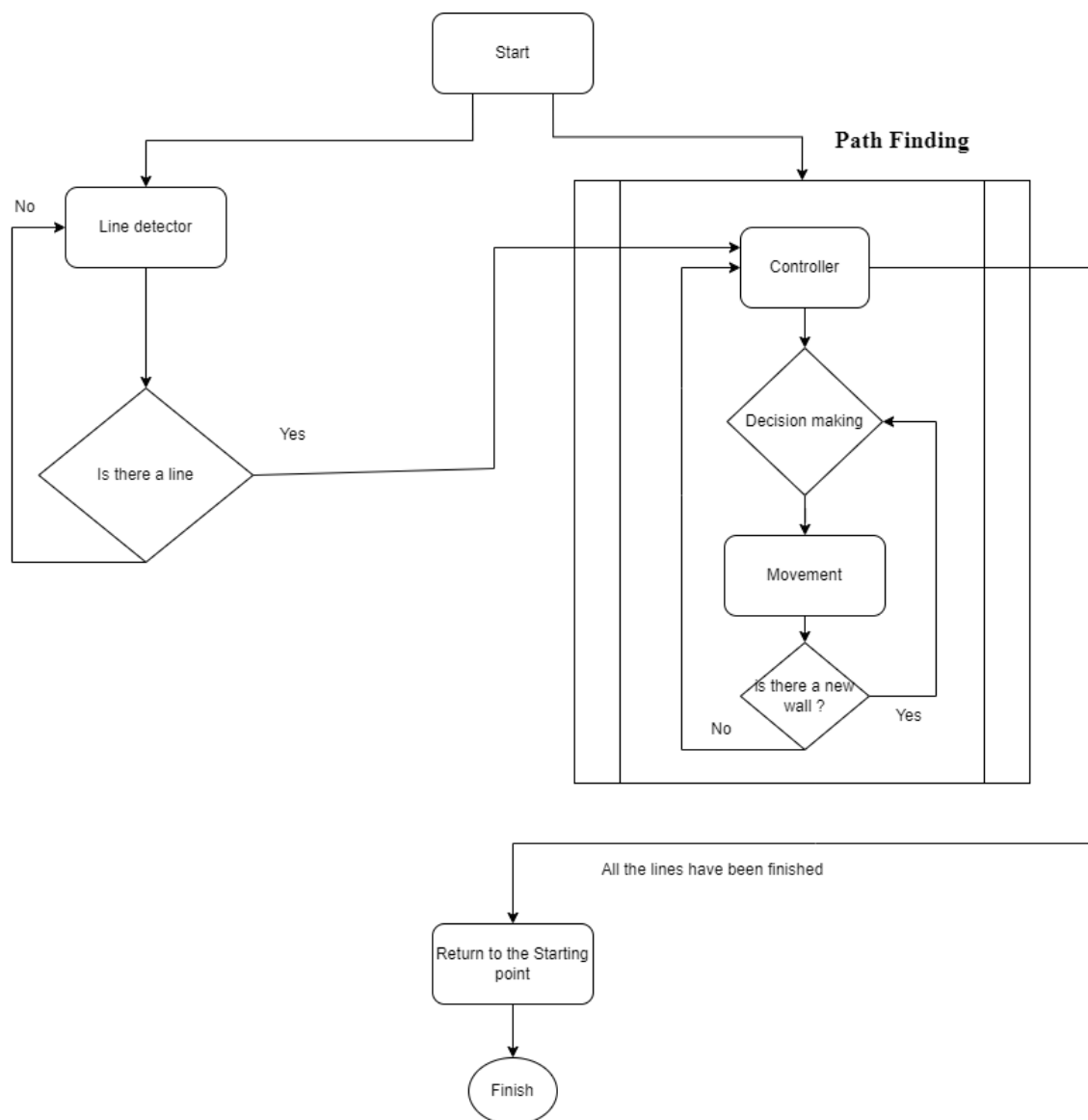
### مقدمه

## مقدمه

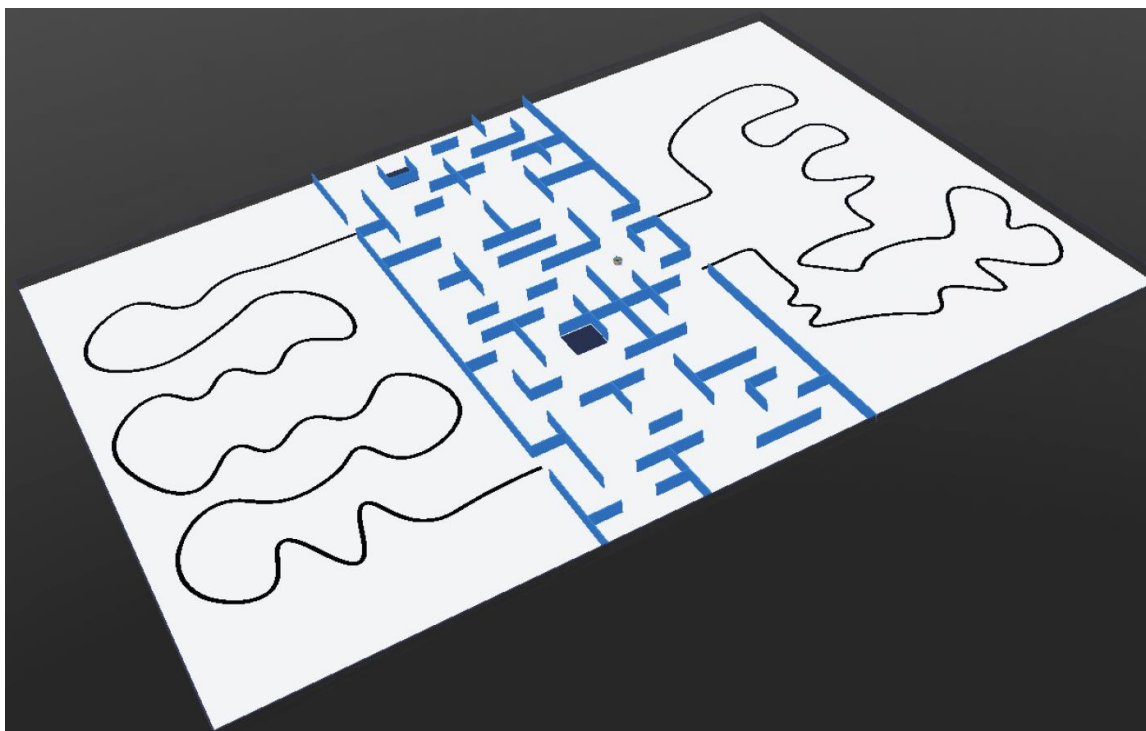
رباتیک به عنوان یکی از پیشروترین شاخه‌های علم و فناوری، نقش مهمی در توسعه سیستم‌های دیجیتال و هوشمند دارد. ربات‌ها امروزه در بسیاری از صنایع و کاربردهای مختلف، از جمله تولید، خدمات، پزشکی و حتی زندگی روزمره ما، نقش مهمی ایفا می‌کنند. یکی از ابزارهای رایج آموزشی و تحقیقاتی در این حوزه، ربات‌های کوچک و 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 بهینه‌ای 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 روش‌های 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 تعیین شده 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 تعیین شده 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 اوایلر 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 مورد استفاده

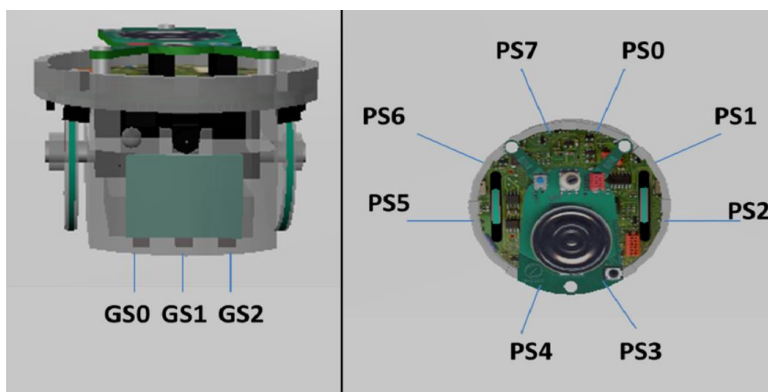
ربات e-puck، ابزار مناسبی برای پیاده‌سازی و تست الگوریتم‌های پیچیده کنترل و مسیریابی در محیط‌های مختلف است. این ربات در این پروژه برای انجام وظایف حل ماز و دنبال کردن خط مورد استفاده قرار گرفته و نتایج عملی به دست آمده از آن، نشان‌دهنده قابلیت‌های بالای این پلتفرم در آموزش و تحقیقات رباتیک است. از ویژگی‌های اصلی ربات e-puck که نکته کلیدی در کدنویسی برای آن است، سنسورهای 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 می‌دهد و به ربات کمک می‌کند تا خطوط را دنبال کند یا تقاطع‌ها را شناسایی کند.
- سنسور مجاورتی: فاصله از اشیاء را اندازه‌گیری می‌کنند و موانع اطراف ربات را تشخیص می‌دهند.
- انکودر: چرخش چرخ‌ها را اندازه‌گیری می‌کنند که می‌توان 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 سنسورهای آن

## فصل دوم

### گام های پیاده سازی

## گام های پیاده سازی

### 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 مورد استفاده 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 می دهیم که حرکت ربات را ذخیره می کند و دارای حافظه برای تشخیص بلوک تکراری می باشد.

بدین ترتیب ربات با الگوریتم DFS، شروع به حرکت و گذر از موانع می کند. همزمان با این پیمایش، اگر ربات خط را تشخیص دهد، اولویت ها تغییر می کنند. ربات با تشخیص اولین خط، ماز را پیمایش می کند تا به سر دیگر خط، متناظر با خط اولیه برسد. سپس آن سر خط را دنبال می کند، 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 می دهد 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

گسسته سازی کنترل کننده PID با استفاده از روش اویلر:

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

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

که در آن T دوره نمونه برداری است.

TM برای ترم مشتق گیری، مشتق را با یک تفاضل تقریب می زنیم:

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

**کنترل کننده PID گسسته:**

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

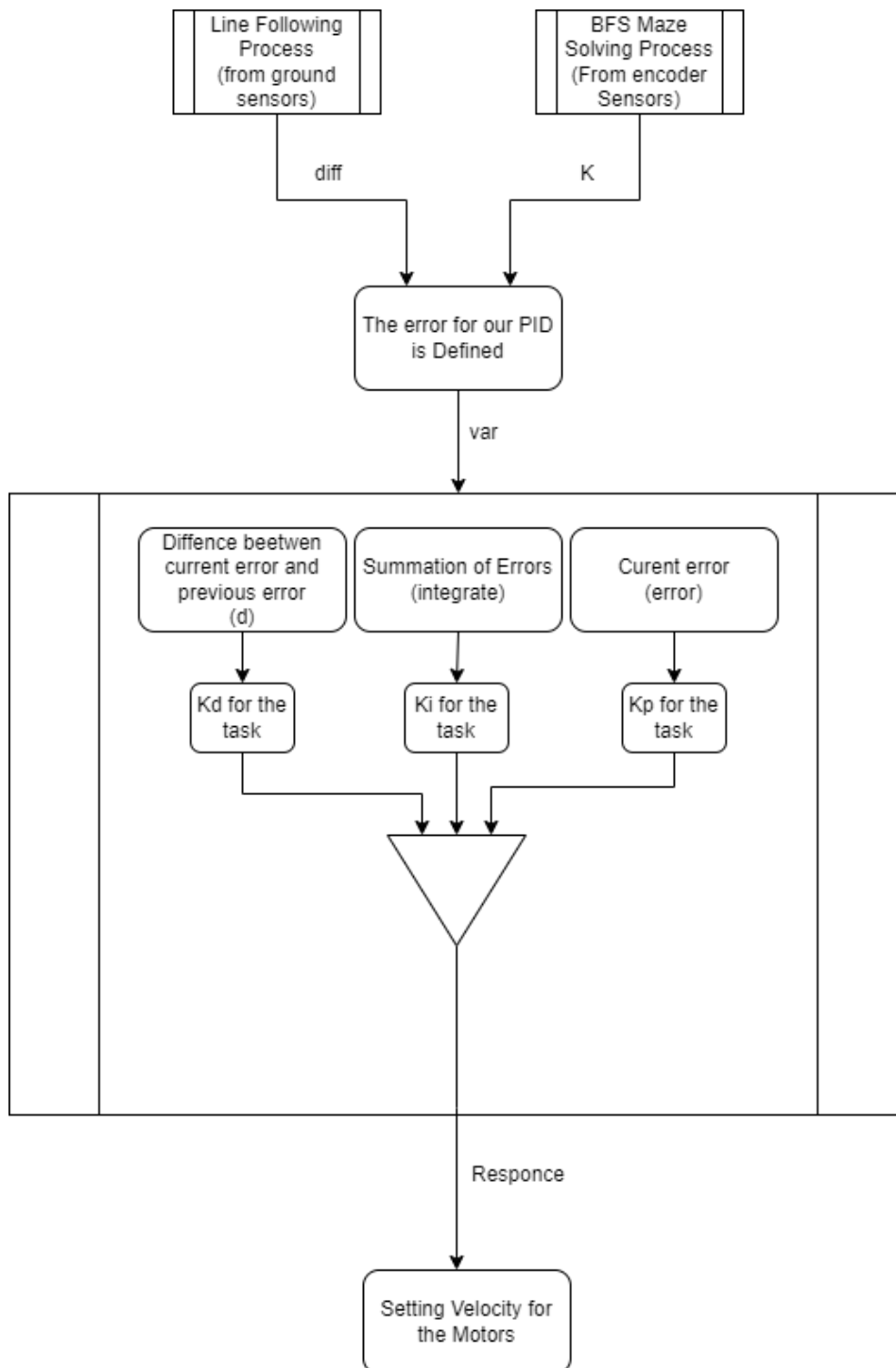
$$u[k] = K_p e[k] + K_i \sum e[k] \cdot T + K_d \frac{e[k] - e[k-1]}{T}$$

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

## PID\_Controller - 2-2-2 کلاس

با `def __init__` مقادیر اولیه کنترلر PID را تنظیم می کنیم.

$K_P$  ضریب تناسبی،  $K_D$  ضریب مشتق گیر و  $K_I$  ضریب انتگرال گیر ما می باشد و با توجه به نحوه ی عملکرد ربات در شبیه سازی 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 می دهیم و در فصل بعدی با جزئیات بیشتر به توضیح کد های شبیه سازی می پردازیم.

در بدنه ی کد، `Error` خطای محاسبه شده در سیکل فعلی، و `pre_error` خطای محاسبه شده در سیکل قبلی اجرای شبیه سازی می باشد. `Integrate` 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 اندازه گیری شده تعریف می کنیم، 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 می آوریم:

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

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

- **ضریب  $k_p$**  بیانگر ضریب تناسبی کنترلر PID می باشد، و افزایش آن سبب بالا رفتن سرعت سیستم می شود که افزایش بالادگی را به همراه دارد که از معایب آن می باشد. این اورشوت در حدی می تواند 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 مشتق گیر کنترلر PID می باشد، و به منظور بهبود پاسخ گذرا و حالت اولیه سیستم به کار می رود. افزایش آن سبب کم شدن نوسانات و جلوگیری از حرکت زیگزاگی می شود، اما در عمل سرعت ربات را کند می کند.
- **ضریب  $k_i$**  بیانگر ضریب انتگرال گیر کنترلر PID می باشد، افزایش آن به بهبود خطای حالت دائم کمک می کند، اما سبب کند شدن سیستم و حتی ایجاد ناپایداری می شود. این پدیده در حرکت چرخشی ربات ایجاد مشکل می کند و می تواند باعث شود ربات در چرخش از مسیر خود منحرف شود.



مقادیر عددی ضرایب PID که با تست شبیه سازی 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 می گیرد. ربات بر مبنای آن، حرکت به جهت دیگری که در اولویت بعدی قرار دارد را اجرا می کند و مسیر خود را پیدا می کند. در این جا از مفهوم جهت مرجع استفاده کرده ایم؛ جهتی که ربات در آن روشن می شود و شروع به حرکت می کند، سمت راست آن، همیشه شمال است. در ابتدا 90 درجه ساعت گرد می چرخد و جهت آن تثبیت می گردد.

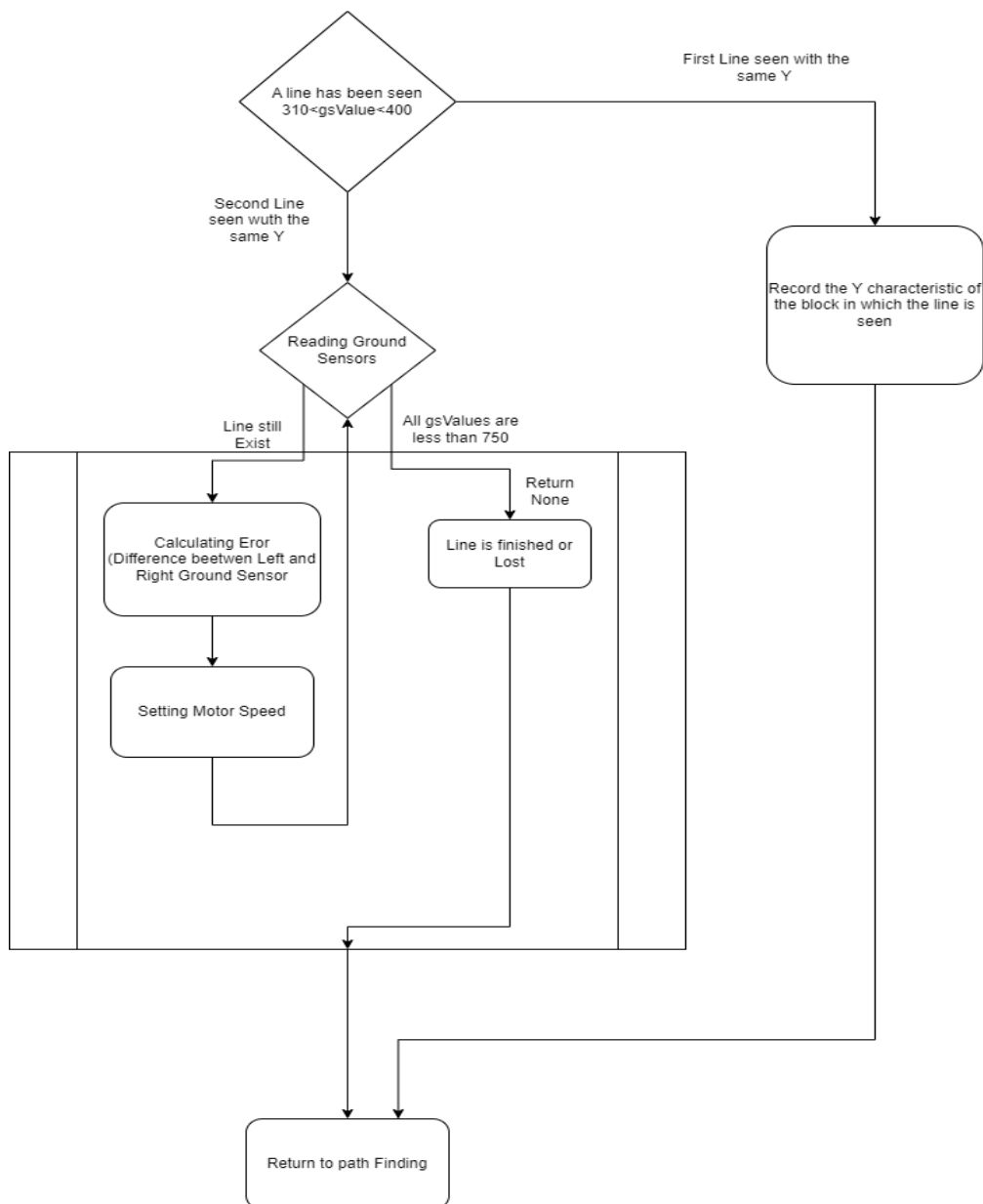
## 2-4- تشخیص خط

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 می کنیم 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 می باشد.

## 2-5- تعقیب خط

میان سه سنسور زمین، ست پوینت ما سنسور وسط است. با توجه به این که کدام یک از سنسور های جانبی و کناره فعال می شوند و خط را می بینند، متوجه انحراف ربات می شویم و بر مبنای آن با اعمال سرعت مناسب به موتورها، ربات را در مسیر اصلی حفظ می نماییم.

**Line Following Block Process**



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 و بخش 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.

▪ کلاس 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.

- کلاس PID\_Controller: این کلاس یک کنترل کننده PID را پیاده سازی می کند که برای تنظیم و کنترل سیستم ها به کار می رود. پارامترهای  $k_p$ ،  $k_i$  و  $k_d$  ضرایب کنترلی هستند و متغیرهایی مانند `error`، `pre_error` و `integrate` برای مدیریت خطاها و محاسبات کنترلی استفاده می شوند.
- تابع `calibr` کالیبراسیون: این تابع به نام `calibr`، خطای اندازه گیری شده توسط سنسورها را کالیبره می کند و جهت حرکت را بر اساس ورودی `direction` تنظیم می کند. با توجه به جهت حرکت ( $f$  برای جلو و  $b$  برای عقب)، سرعت موتورهای تنظیم می شود تا خطا به حد مطلوب برسد.
- تابع `read_encoder`: این تابع به نام `read_encoder`، مقادیر انکودرها را می خواند. با استفاده از یک حلقه، مقادیر دو انکودر را در لیستی جمع آوری کرده و در نهایت آن را برمی گرداند.
- تابع `move_func`: این تابع به نام `move_func` برای حرکت و چرخش ربات استفاده می شود. پارامترهای `block_rate` برای میزان حرکت، `rotation` برای زاویه چرخش و `direction` برای جهت چرخش (ساعت گرد یا پادساعت گرد) استفاده می شوند.
- ابتدا، مقادیر انکودرها خوانده می شوند. در صورت نیاز به چرخش، زاویه به رادیان تبدیل شده و سپس ربات با استفاده از کنترل کننده PID چرخش را انجام می دهد.
- در مرحله بعد، حرکت به سمت جلو یا عقب با استفاده از کنترل کننده PID انجام می شود تا میزان حرکت مورد نظر طی شود. در هر مرحله، مقادیر انکودر به روز شده و سرعت موتورهای تنظیم می شود.
- تابع `get_gs_values`: این تابع به نام `get_gs_values`، مقادیر سنسورهای زمین (`gs`) را می خواند. با استفاده از یک حلقه، مقادیر سه سنسور را در یک لیست جمع آوری کرده و در نهایت آن را برمی گرداند.

- 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 دیاگرام 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.



○ مازول controller: این مازول که 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 مازول movement\_controller ارسال نماید. در این مازول سعی شده تا با در نظر داشتن اطلاعاتی که از قبل کسب کردیم و همچنین اهدافی که داریم بتوانیم بهینه ترین روش در جست و جوی نقشه را پیاده سازی کنیم.

به صورت هم زمان در این مازول با خواندن فلگ line\_flag، دستورات مربوطه به این پخش اجرا می شود 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 کرده ایم آن خط را پیدا کنیم

لـ همچنین در این مازول 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 پیاده سازی کرده ایم و در صورتی که محور Y ما که یعد از خارج شدن از خط ممکن است به صورت کامل از حالت کالیبر در بیاید، کالیبر نشود، از خانه ی والد به منظور کالیبر کردن محور عمودی کمک می گیریم و در ادامه فرآیند جست و جوی نقشه را ادامه می دهیم تا زمانی که همه ی خطها را بپیماییم، این نکته حائز اهمیت می باشد که در تمامی عملیات های این مازول اولویت های بررسی به صورت دینامیکی طوری تنظیم شده تا بهینه ترین جست و جو را برای یافتن خطوط داشته باشیم و همچنین بتوانیم استراتژی اصلی خود که وارد نشدن به سر اول خط می باشد را پیاده سازی کنیم، در تمامی عملیات ها در صورت نیاز کالیبراسیون ربات انجام گیرد.

• تابع back\_to\_home: زمانی که تمامی خطهای مورد نیاز را طی کردیم، این متد اجرا می شود تا ربات از طریق نزدیک ترین مسیری که طی کرده بود، به نقطه ی شروع باز گردد.

• تابع show\_tree: در نهایت، این تابع برای نمایش ساختار درختی نقشه استفاده می شود.

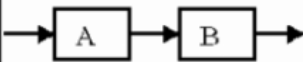
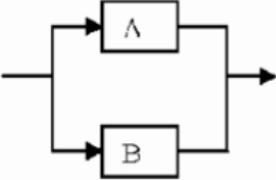
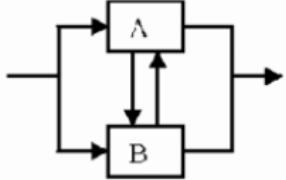
## فصل چهارم

### چالش های مطرح شده و راه حل آن

## چالش های مطرح شده و راه حل آن

### Task Dependency -1-4

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

Three Types of Task Dependencies			
Relationship	Sequential (Dependent)	Parallel (Independent)	Coupled (interdependent)
Graph Representation			

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

انواع وابستگی تسک ها و مدیریت آن:

### 1. تسک های ترتیبی (dependent)

این تسک ها وابسته به تکمیل تسک های قبلی هستند. در کد، این کار با ترتیب دهی صریح در داخل توابع و استفاده از تأخیرها برای اطمینان از تکمیل یک تسک قبل از شروع تسک بعدی مدیریت می شود. ✓ تابع `move_func` ربات را حرکت می دهد و منتظر می ماند تا حرکت کامل شود قبل از اینکه اجازه دهد تسک بعدی شروع شود.

✓ تابع `delay` برای ایجاد تأخیر زمانی بر اساس زمان بندی ربات استفاده می شود تا اطمینان حاصل شود که تسک ها به ترتیب تکمیل می شوند.

### 2. تسک های موازی (independent)

این تسک ها می توانند به صورت موازی اجرا شوند بدون اینکه به یکدیگر وابسته باشند. در کد، از `threading` برای مدیریت چنین تسک هایی استفاده می شود. ✓ `line_detector` و `controller` به صورت موازی در `thread` های جداگانه اجرا می شوند.

### 3. تسک های وابسته و متصل (Coupled)

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

✓ داده های مشترک و همگام سازی: متغیرهای مشترک مثل `line_flag` و `line_flag_write` برای هماهنگی بین بخش های مختلف کد استفاده می شوند. متغیر `line_flag` نشان می دهد که آیا ربات خطی را شناسایی کرده است یا خیر که بر تصمیمات حرکتی آن تأثیر می گذارد.

## Synchronizing -2-4

همان گونه 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 می کنیم.

همگام سازی (Synchronizing) به فرآیند هماهنگ سازی اجرای چندین تسک به منظور جلوگیری از مشکلاتی مانند شرایط مسابقه (race conditions) و حفظ یکپارچگی داده ها اشاره دارد. همگام سازی تضمین می کند که منابع مشترک به درستی مدیریت می شوند و تسک ها بدون تداخل با یکدیگر اجرا می شوند. در این کد از thread برای اجرای موازی توابع controller و line\_detector استفاده شده است. این تردها به طور همزمان اجرا می شوند اما برای همگام سازی از join برای منتظر ماندن تا تکمیل هر ترد استفاده شده است.

✓ متغیرهایی مانند line\_flag و line\_flag\_write برای هماهنگی بین تسک های مختلف استفاده می شوند. این متغیرها وضعیت های مختلف را ذخیره می کنند که تسک های مختلف می توانند بر اساس آن ها تصمیم گیری کنند.

✓ در مواردی که نیاز به محافظت از منابع مشترک و جلوگیری از تداخل همزمان وجود دارد، از قفل ها استفاده می شود. در این کد، می توان از threading.Lock برای ایجاد قفل و همگام سازی دسترسی به متغیرهای مشترک استفاده کرد (هرچند در کد فعلی قفل ها به طور صریح استفاده نشده اند، اما این یک روش رایج برای همگام سازی است).

```
lock = threading.Lock()
```

```
with lock:
```

```
# critical section
```

```
line_flag = True
```

## Calibration -3-4

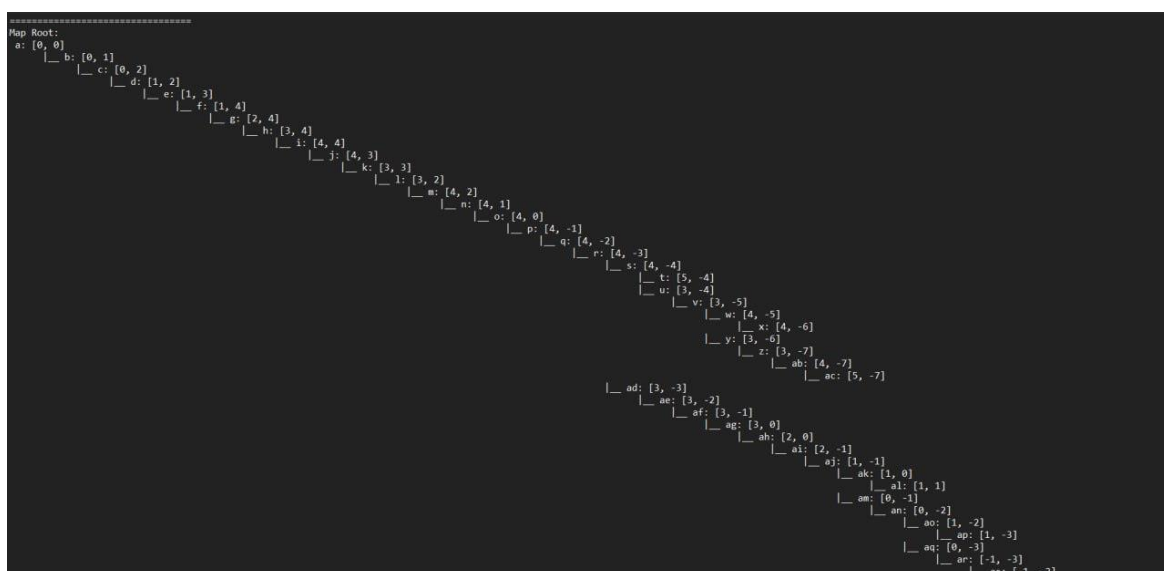
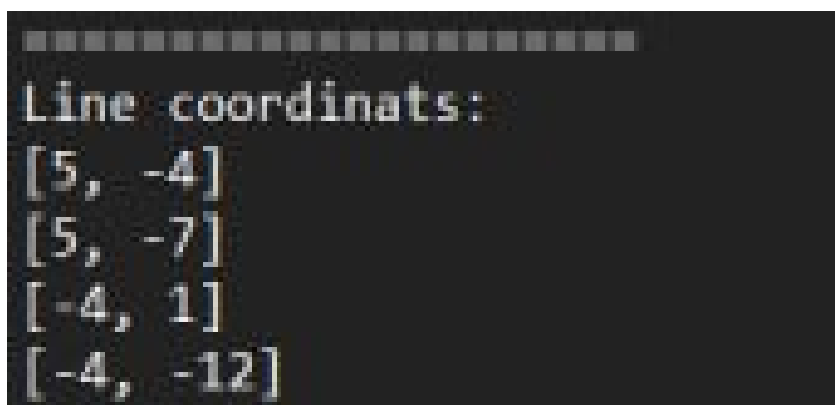
ربات در حرکت خود، چه به صورت چرخشی چه حرکت مستقیم به سوی جلو یا عقب، نیاز به دقت بالا دارد که از مسیر اصلی خود خارج نشود و الگوریتم به خوبی اجرا شود. این دقت در جابجایی تا حد خوبی به وسیله کنترلر PID تضمین شده است؛ هرچند برای بهینه ترین شبیه سازی همچنان نیاز به کالیبراسیون داریم. کالیبراسیون با استفاده از دیوار های اطراف بلوک انجام می گیرد. ربات اول به دنبال دیوار می گردد 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 می شود 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 طراحی شده را در محیط شبیه‌سازی webots پیاده سازی کردیم و نتیجه مطلوب را از آن گرفتیم. ویدیوی این شبیه‌سازی در فایل پروژه پیوست شده‌است.



شکل 5-1- چاپ خروجی در بخش console



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

- همانطور که در شبیه‌سازی 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 مسیرهای طی شده به هر بلوک و همچنین مختصات ورودی‌های هر خط را در خروجی چاپ کند.
- با تست‌های عملی متنوع و تغییر اولویت‌های گردش به صورت دینامیک توانستیم زمان اجرا را تا 60 درصد کاهش دهیم که بهینه‌سازی بسیار مناسبی برای این ربات با این مجموعه تسک‌ها می‌باشد.
- با ایجاد دو روش کالیبراسیون توانستیم خطای مکانی را تا 95 درصد و خطای زاویه‌ای را تا 99 درصد در هر بلوک کاهش دهیم و همچنین با ایجاد سیستم کالیبراسیون بعد خطوط توانستیم خطای زاویه‌ای را تا 99 درصد و خطای مکانی را تا 93 درصد کاهش دهیم.
- با موازی‌سازی دو تابع `line_detector` و `controller` کار تشخیص خط را بسیار ساده‌تر کردیم به این دلیل که دیگر لزومی به ایجاد توقف‌های زمانی و در نظر گرفتن این توقف‌ها در مراحل مختلف فرایند‌های تصمیم‌گیری ربات نبوده است.
- این نکته حائز اهمیت می‌باشد که این ربات می‌تواند با قرار گرفتن در هر نقطه‌ای از نقشه به شرطی که در مرکز بلوک خود باشد و با هر تغییر زاویه‌ای به شرطی که اختلاف آن با زاویه فعلی مضربی از 180 درجه باشد نیازمندی‌ها را ارضا کند.
- در برخی شرایط خاص با توجه به نحوه قرارگیری ربات ممکن است بهینه‌سازی که در مورد دوم به آن اشاره کردیم به ثمر نرسد، اما در اکثر حالات این تغییر اولویت جست و جو به صورت دینامیک باعث بهبود زمان عملکرد ربات می‌شود.
- با رصد کردن عدد خوانده شده از انکدرهای 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 موتورها مواجه هستیم که همین دلیل سبب ایجاد خطا در بلند مدت می‌گردد.

### پیشنهادهای:

- ✓ استفاده از GPS برای تسهیل و تسریع سیستم مسیریابی و حرکتی ربات در ماز.
- ✓ استفاده از ماشین لرنینگ برای بهبود الگوریتم یافتن ورودی‌های مختلف خط. همچنین میتوانیم از روش reinforcement learning در این بخش استفاده کنیم.
- ✓ استفاده از چند PID مختلف برای رنج‌های مختلف حرکتی و حرکت‌های مختلف تا بتوانیم حرکت ربات را تسریع بخشیم و خطا را به حداقل برسانیم.
- ✓ برای بهبود عملکرد ربات بطوری که در هر نقطه‌ای و در هر پوزیشنی بتواند به بهترین شکل نقشه را بپیماید میتوانم چهار تسک الگوریتم حرکت کردن، تشخیص خط، خواندن انگشت موتور، و تشخیص دیوار را موازی‌سازی 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>