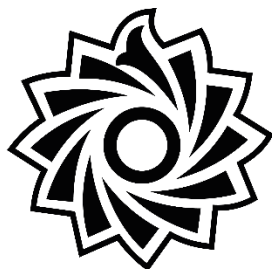


بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



دانشگاه تربیت مدرس

پایان نامه دوره کارشناسی

مهندسی برق - الکترونیک

عنوان پروژه:

مشارکت ربات ها در حل مسئله ماز

استاد راهنما:

دکتر منصور باقری

دانشجویان:

محسن رحیمی و امیرحسین رحمانی

بهمن ماه ۱۴۰۰

تشکر و قدردانی:

از همه کسانی که در این پایان نامه ما را یاری رساندند بخصوص دکتر باقری کمال تشکر را داریم.

چکیده:

توسعه سریع تکنولوژی‌ها ما را به برنامه ریزی دقیق برای بهترین انتخاب سوق می‌دهد. این فناوری و نوآوری‌ها باعث این شده است که کار افراد را در زندگی آسان تر کند. در این مقاله، ربات حل ماز با مهارت نقشه برداری و بومی سازی مستقل توسعه یافته است. درگام اول، ربات حل ماز با سه حسگر فاصله سنج اولتراسونیک طراحی شده است که برای تشخیص دیوار برای جلوگیری از برخورد و برای تشخیص موانع استفاده می‌شود. همچنین انتظار می‌رود از ربات در محیطی که برای انسان غیرقابل دسترس است استفاده شود. علاوه بر این، مکان‌هایی نیز وجود دارد که استفاده از ربات‌ها تنها راه رسیدن به هدف است. ما با موفقیت توانایی حل ماز را بر روی ربات پیاده سازی کرده ایم. نتیجه آزمایش به این صورت بود که ربات می‌تواند ماز را با موفقیت بدون برخورد با دیوارها حل کند و همچنین مسیر صحیح را در حافظه خود ذخیره کند و در انتهای مسیر یعنی موقعی که به هدف رسید مسیر ذخیره شده را برای ربات همکار ارسال کند. در این طراحی، امکان ارسال موقعیت به ربات‌های دیگر تعبیه شده هست.

کلید واژه‌ها:

ماز- میکروماوس- ربات حل ماز - الگوریتم - گراف - پیمایش گراف - میکروکنترلر - ATmega328 -

سنسور ها

مقدمه

در این پروژه هدف پیاده سازی الگوریتم برای حل ماز و پیدا کردن مسیر صحیح و ارسال آن به ربات هدف است که در پنج فصل تهیه شده است.

فصل اول :

در فصل اول مقدمه ای از ربات های ماز تاریخچه و همچنین الگوریتم های مختلف و معروف برای حل آن پرداخته شده است.

فصل دوم :

در این فصل راجع به خواست پروژه و هدف آن توضیح داده شده است.

فصل سوم :

در این فصل که تحت عنوان تئوری سازی تنظیم شده راجع به انتخاب الگوریتم برای حل پروژه صحبت می شود.

فصل چهارم :

شبیه سازی: پیاده سازی الگوریتم بر روی میکرو ATmega328 و پلتفرم Arduino

فصل پنجم :

پیاده سازی در عمل: پیاده سازی تست الگوریتم نوشته شده در محیط واقعی آخرین فصل را تشکیل می دهد.

فهرست عناوین

فصل اول – مقدمه ای بر ربات های ماز	۱
۱-۱- تاریخچه	۲
۲-۱- مقدمه ای بر ربات های حل ماز	۲
۳-۱- الگوریتم های حل ماز	۴
۴-۱- کارایی	۴
فصل دوم – خواست و هدف پروژه	۵
فصل سوم-تئوری سازی	۸
۱-۳- انتخاب مسیر	۹
۲-۳- انتخاب الگوریتم برای حل ماز	۱۰
۳-۳- پیمایش در گراف ها و نحوه پیاده سازی آن ها	۱۱
۱-۳-۳- جستجوی اول سطح	۱۱
۲-۳-۳- پیمایش عمق اول	۱۵
۴-۳- الگوریتم های معروف و نحوه پیاده سازی آن ها	۱۷
۱-۴-۳- الگوریتم تصادفی ماوس	۱۷
۲-۴-۳- الگوریتم دنبال کردن دیواره ها	۱۸
۳-۴-۳- الگوریتم pledge	۱۸
۴-۴-۳- الگوریتم Trémaux's	۱۹
۵-۴-۳- Dead-end filling	۲۱
۶-۴-۳- الگوریتم بازگشتی	۲۱
۷-۴-۳- الگوریتم کوتاه ترین مسیر	۲۲
۸-۴-۳- Flood Fill	۲۳

۲۷	۳-۵- روش های ارتباطی بین دو دستگاه
۲۷	۳-۶- انتقال بی سیم
۲۸	۳-۶-۱- انتقال رادیویی
۲۸	۳-۶-۲- انتقال میکروویو
۲۹	۳-۶-۳- انتقال مادون قرمز
۳۰	۳-۶-۴- انتقال نوری
۳۲	فصل چهارم- ابزارها و راه اندازی آن ها
۳۰	۴-۱- معرفی میکروکنترلر
۳۱	۴-۲- انتخاب سنسور ها و نحوه راه اندازی آن ها
۳۲	۴-۳- سنسور فاصله سنج اولتراسونیک
۳۷	۴-۳-۱- طرز کار اولتراسونیک
۳۷	۴-۳-۲- نحوه ایجاد موج فراصوتی
۳۸	۴-۳-۳- نحوه عملکرد pulseIn
۳۸	۴-۳-۴- کد نوشته شده برای اندازه گیری مسافت سنسور تا مانع در آردواینو
۴۰	۴-۴- انتخاب موتور
۴۵	۴-۵- PID کنترلر چیست؟
۴۶	۴-۵-۱- پیاده سازی PID کنترلر با استفاده از Arduino
۴۷	۴-۵-۲- کد نوشته شده در نرم افزار Arduino برای پیاده سازی PID کنترلر
۴۸	۴-۶- درایور موتور
۴۹	۴-۶-۱- درایورهای مختلف موتور برای ربات های خود ران
۵۰	۴-۷- مازول درایور موتور L298N و راه اندازی آن
۵۰	۴-۷-۱- راه اندازی درایور L298N با آردوینو
۵۱	۴-۷-۲- کد نوشته شده برای راه اندازی مازول Lm298N

۵۳	۸-۴- مازول درایور موتور L9110S و راه‌اندازی آن
۵۴	۴-۸-۱- راه‌اندازی مازول درایور L9110S با آردوینو
۵۴	۴-۸-۲- کد نوشته شده برای راه‌اندازی مازول L9110S
۵۶	۴-۹- مازول وایرلس nRF24L01
۵۷	۴-۹-۱- راه‌اندازی مازول nRF24L01
۵۷	۴-۹-۲- کد نوشته شده برای راه‌اندازی مازول nRF24L01
۵۹	۴-۱۰- انتخاب منبع تغذیه
۶۰	۴-۱۱- پارامترهای عملکرد باتری
۶۵	فصل پنجم- پیاده‌سازی در عمل
۶۶	۵-۱- ساخت مسیر
۶۷	۵-۲- ساخت ربات
۶۹	۵-۳- الگوریتم نوشته شده در نرم افزار Arduino
۷۱	۵-۴- نتایج به‌دست آمده
۷۱	۵-۵- کارهایی که در ادامه می‌توان انجام داد
۷۲	پیوست ۱
۹۲	منابع

فهرست تصاویر

شکل ۱-۱: مسیر ماز	۲
شکل ۲-۱: ربات میکرو ماوس	۳
شکل ۱-۳: نمونه از مسیر های ماز	۹
شکل ۲-۳: نمونه از مسیر های ماز	۹
شکل ۳-۳: مسیر ماز انتخاب شده برای حل مسئله ماز	۱۰
شکل ۴-۳: نظریه گراف	۱۱
شکل ۵-۳: جستجوی گراف براساس روش BFS	۱۱
شکل ۶-۳: جستجوی گراف براساس BFS	۱۲
شکل ۷-۳: نمونه‌ی عملی از پیمایش گراف براساس الگوریتم BFS	۱۴
شکل ۸-۳: جستجوی گراف براساس DFS	۱۴
شکل ۹-۳: نمونه‌ی عملی از پیمایش گراف براساس DFS	۱۶
شکل ۱۰-۳: الگوریتم دنبال کردن دیواره‌ها	۱۷
شکل ۱۱-۳: الگوریتم PLEDGE	۱۷
شکل ۱۲-۳: طیف‌های الکترومغناطیسی	۲۵
شکل ۱۳-۳: انتقال رادیویی	۲۵
شکل ۱۴-۳: انتقال مایکروویو	۲۶
شکل ۱۵-۳: انتقال نوری	۲۷
شکل ۱۶-۳: ماژول وایرلس nRF24L01	۲۸
شکل ۱-۴: آردوینو مدل اونو	۳۰
شکل ۲-۴: سنسور شدت نور	۳۲
شکل ۳-۴: طرز کار سنسور اولتراسونیک	۳۶
شکل ۴-۴: طرز کار سنسور اولتراسونیک	۳۷

- شکل ۴-۵: مدار راهاندازی سنسور شدت نور ۳۷
- شکل ۴-۶: موتور گیربکسی پلاستیکی ۴۴
- شکل ۴-۷: یک سیستم فیدبک با کنترل کننده PID ۴۵
- شکل ۴-۸: کنترل موتور ها از طریق PWM با PID کنترلر ۴۶
- شکل ۴-۹: ماژول درایور L298N ۵۰
- شکل ۴-۱۰: مدار راهانداز موتور DC به وسیله ماژول Lm298N و آردوینو در نرم افزار پروتئوس .. ۵۰
- شکل ۴-۱۱: ماژول درایور L9110S ۵۳
- شکل ۴-۱۲: مدار راه انداز ۲ موتور DC به وسیله درایور L9110S با آردوینو ۵۴
- شکل ۴-۱۳: شماتیک nRF24L01 برای خروجی ۵۰ اهم RF ۵۴
- شکل ۴-۱۴: نحوه راهاندازی ماژول nRF24L01 با Arduino ۵۷
- شکل ۵-۱: پیاده سازی ماز طراحی شده ۶۶
- شکل ۵-۲: پیاده سازی نهایی ربات در محیط واقعی نمونه ۴ چرخ ۶۸
- شکل ۵-۳: پیاده سازی نهایی ربات در محیط واقعی نمونه ۳ چرخ ۶۸
- شکل ۵-۴: محیط نرم افزار Arduino ۶۹

فصل اول

مقدمه‌ای بر ربات های ماز

۱-۱- تاریخچه ربات های حل ماز (میکروماوس)

میکروماوس^۱ رویدادی است که در آن ربات های کوچک ماز ۱۶*۱۶ را حل می کنند. که از اواخر

دهه ۱۹۷۰ آغاز شد. رویدادها در سراسر جهان برگزار می شوند و در بریتانیا، ایالات متحده،

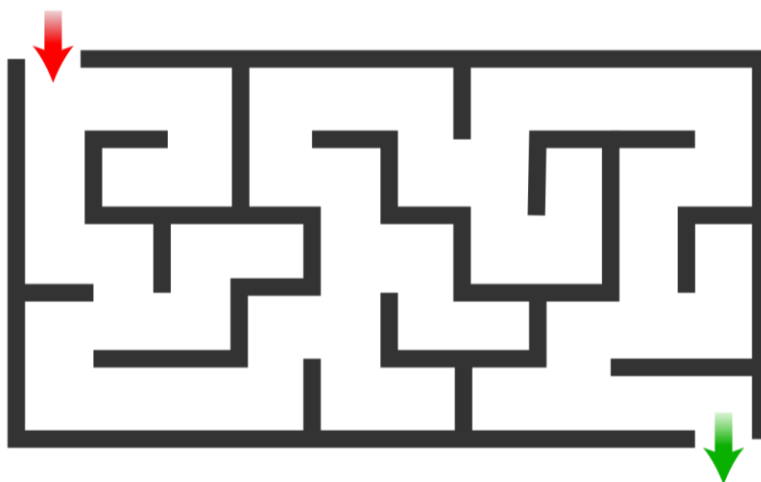
ژاپن، سنگاپور، هند، کره جنوبی محبوبیت بیشتری دارند و در کشور های شبه قاره مانند سریلانکا محبوب هست.

۱-۲- مقدمه ای بر ربات های حل ماز

ربات های ماز یا میکرو ماوس ها گروهی از ربات ها هستند که می توانند یک مسیر ناشناخته را طی

کنند و به مقصد برسند. مسیر ناشناخته در مسابقات و آزمایش ها یک ماز است و در عمل می تواند یک شهر یا یک مکان نامشخص باشد.

در زیر تصویر یک نمونه ماز را مشاهده میکنید.



شکل ۱-۱: مسیر ماز

همچنین در تصویر زیر نمونه ای ربات حل ماز را مشاهده می کنید.

¹ Micromouse



شکل ۱-۲: ربات میکروماوس [1]

ماز از سلول ها و شبکه های ۱۶*۱۶ تشکیل شده است که هرکدام ۱۸۰ میلی متر مربع با دیوارهایی به ارتفاع ۵۰ میلی متر است. ربات های ماز، ربات های کاملاً مستقلی هستند که باید بدون کمک راه خود را از یک موقعیت شروع از پیش تعیین شده به سمت نقطه مرکزی ماز پیدا کنند.

ربات های ماز باید مکان خود را رهگیری کنند، دیواره ها را در حین کاوش کشف کنند، ماز را ترسیم کنند و تشخیص دهد که چه زمانی به هدف رسیده است. پس از رسیدن به هدف، ماوس آن مسیر را در کمترین زمان ممکن اجرا خواهد کرد.

مسابقات میکرو ماوس تحت عنوان ³IEEE، ²UK و کنفرانس ها تحت عنوان ⁴Minos به طور

منظم برگزار می شوند. [1]

² UK Micromouse and Robotics Society

³ Micromouse USA - USA Micromouse Fans Sit

⁴ Minos - UK Micromouse and Robotics Society

میکرو ماوس نیم سایز - Half-Size Micromouse

نسخه جدیدی از میکرو ماوس که نیم سایز نامیده می‌شود که در ۳۰ امین مسابقات میکرو ماوس در سال ۲۰۰۹ در ژاپن با عنوان robolaboN معرفی شد. در نسخه جدید بجای ماز ۱۶*۱۶، در مسابقات جدید از ماز ۳۲*۳۲ استفاده شد. که ابعاد سلول ها و دیوار ها به نصف کاهش پیدا کرد.[1]

۱-۳- الگوریتم های حل ماز

ماوس ها می‌توانند از الگوریتم های مختلف جستجو استفاده کنند. الگوریتم های جستجو متداول از جمله روش Bellman flood-fill ، Dijkstra's، الگوریتم معروف جست و جو معروف و صنعتی A^* در میان الگوریتم های مختلف پیمایش گراف ها و پیمایش درخت استفاده کرد.[1]

۱-۴- کارایی

ربات های ماز یا همان ماوس ها بسته به طرح های ماز می‌توانند با سرعت بیش از سه متر در ثانیه حرکت کنند. برخی از بهترین سازندگان میکروموس عبارتند از: یوسوکه کاتو، نگ بنگ کیات و فومیتا کا ناکاشیما هستند. رکورد جهانی کنونی ۳.۹۲۱ ثانیه و در اختیار نگ بنگ کیات است.[1]

عملکرد ربات ها در سال های اخیر به طور قابل توجهی بهبود یافته است. از سال ۲۰۱۵، ماوس های برنده احتمالا با شتاب رو به جلو و ترمز بالای ۱۰ متر بر ثانیه حرکت کنند. Micromice از جمله ربات های خودران با بالاترین عملکرد هستند.

فصل دوم

خواست و هدف پروژه

خواست پروژه:

عنوان پروژه: مشارکت ربات ها برای حل مسئله ماز

مشخصات کیفی، بلوک دیاگرام پروژه :

مسئله تعریف شده به این شکل است که دو ربات وجود دارد ۱- ربات اصلی^۵ ۲- ربات همکار^۶

مسئله تعریف به یکی از دو شکل زیر میتواند انجام شود:

شکل اول:

ربات اصلی باید موقعیت مکانی خود را توسط سنسورها نسبت به موانع بخواند، سپس موقعیت خود را به ربات همکار ارسال کند ربات ها باید به کمک هم ماز از پیش تعیین شده را حل کنند و به مسیر انتهایی برسند.

شکل دوم:

ربات اصلی ابتدا مسیر درست ماز را باید با بهره گیری از هوش خود پیدا کند سپس مسیر درست را برای ربات همکار ارسال کند تا ربات همکار در کوتاه ترین زمان مسیر صحیح را بییماید و به مقصد تعیین شده برسد.

شرح پروژه :

ربات اصلی با توجه به سنسور هایی که باید برای آن انتخاب شود موقعیت خود را نسبت به موانع میخواند سپس با توجه به الگوریتم انتخابی شروع به حرکت در ماز تعیین شده میکند بعد از کاوش کردن در ماز باید مسیر صحیح را ذخیره کند هم چنین در طول مسیر باید بتواند بر روی موتور های خود کنترل داشته

⁵ Main Robot

⁶ Fellow Robot

باشد تا با شتاب مناسب و ترمز های به موقع بتواند ماز را حل کند و در انتها باید بتواند با ربات همکار ارتباط برقرار کند که برای ارتباط با ربات همکار میتوانیم از ماژول های وایفای و وایرلس برای برقراری این ارتباط استفاده کنیم که ربات اصلی باید مسیر صحیح ذخیره شده بعد رسیدن به مقصد نهایی برای ربات همکار ارسال کند، سپس ربات همکار بعد از دریافت اطلاعات باید شروع به پیمودن مسیری که ربات اصلی فرستاده میکند تا به مقصد هدف برسد.

مراحل و فازهای انجام پروژه:

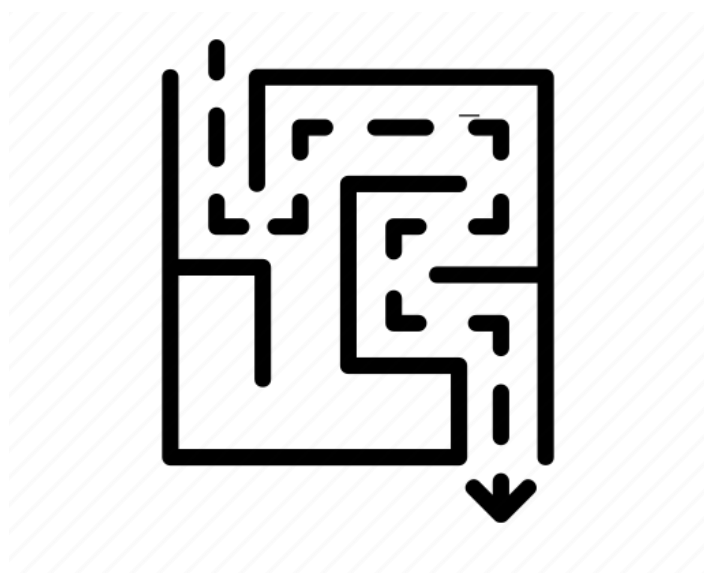
در ابتدا باید مسیر ماز طراحی و انتخاب شود، در مرحله بعد باید انتخاب شود با چه تکنولوژی و الگوریتم و ابزاری قرار است کار کنیم (برای تصمیم گیری در این باره باید به میزان بودجه و وقت خود نگاه کنیم) بعد از تصمیم گیری در این باره باید، ابزارهای لازم و مناسب تهیه شوند. سپس باید ربات طراحی شود که با توجه به مسیری که طراحی شده ابعاد ربات و طراحی آن صورت می گیرد. بعد از این مراحل به سراغ پیاده سازی الگوریتم بر روی تکنولوژی انتخابی می رویم بعد از پیاده سازی الگوریتم و طراحی ربات ها در مرحله آخر باید به سراغ پیاده سازی آن در محیط واقعی برویم بعد از انجام مرحله نتیجه نهایی باید در قالب گزارش پایانی گردآوری و تهیه شود.

فصل سوم

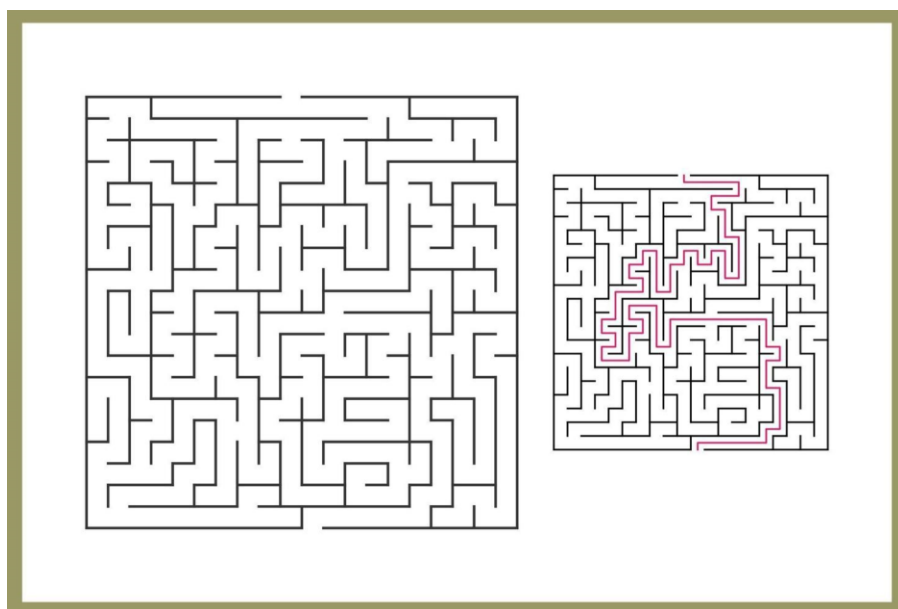
تئوری سازی

۳-۱- انتخاب مسیر

ماز ها میتوانند مسیرهای بسیار پیچیده یا مسیر های ساده باشند که در زیر چند نمونه از مسیر های ماز آورده شده است.



شکل ۳-۱: نمونه از مسیر های ماز

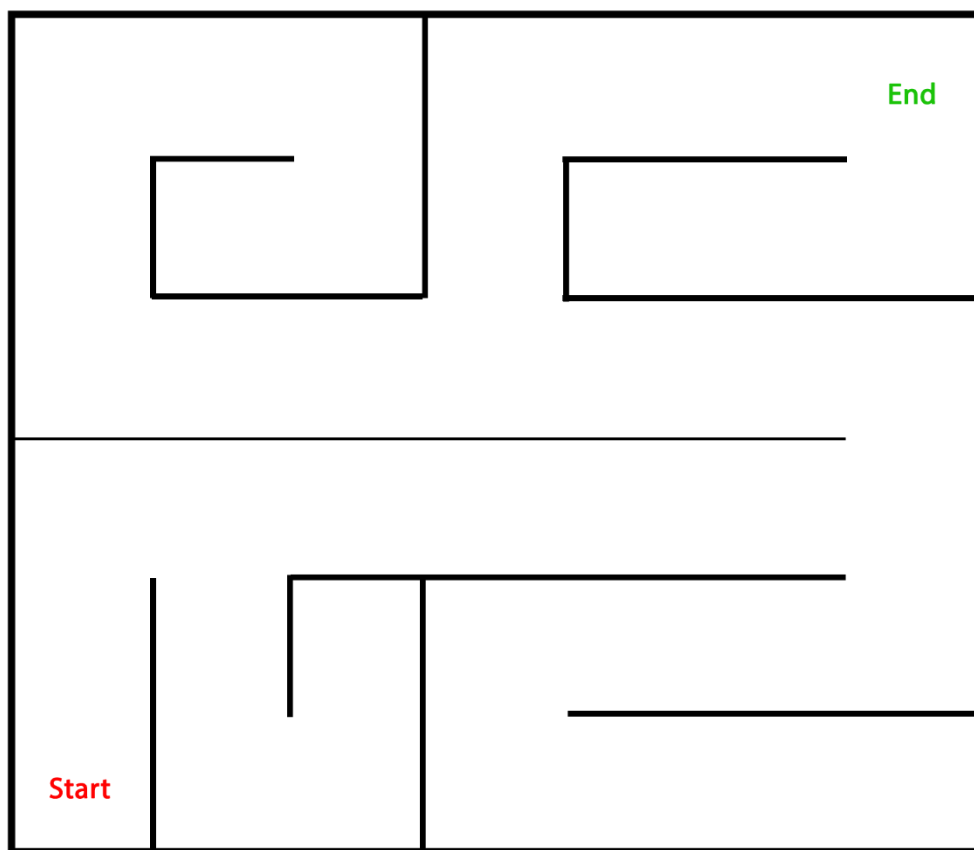


شکل ۳-۲: نمونه از مسیر های ماز

برای این پروژه ماز زیر را به عنوان مسیر انتخابی برای پروژه خود انتخاب شد :

که همانطور که مشاهده میکنید یک ماز 6×6 است. که باید بر اساس مسیر انتخابی و خواسته پروژه،

الگوریتم انتخاب و طراحی شود.



شکل ۳-۳: مسیر ماز انتخاب شده برای حل مسئله ماز

۳-۲- انتخاب الگوریتم برای حل ماز

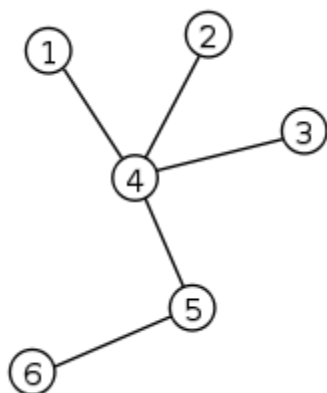
همانطور که در فصل اول صحبت شد برای حل ماز الگوریتم های زیادی وجود دارد که چند تا از

معروف ترین الگوریتم های حل ماز را در قسمت اول نام برده شد و حالا در این قسمت مفصل تر راجع به

این الگوریتم ها و هم چنین الگوریتم های پیمایش گراف ها و درخت ها صحبت می شود.

الگوریتم های حل ماز ارتباط نزدیکی با نظریه گراف ها دارند. اگر بتوان مسیر های ماز به روش مناسب بکشد می توان نتیجه را شبیه درخت کرد.

درخت: در نظریه گراف، درخت یک گراف بدون جهت است که در آن هر دو راس توسط یک مسیر به هم متصل شده است یا به به طور معادل یک گراف همبند و بدون دور است.

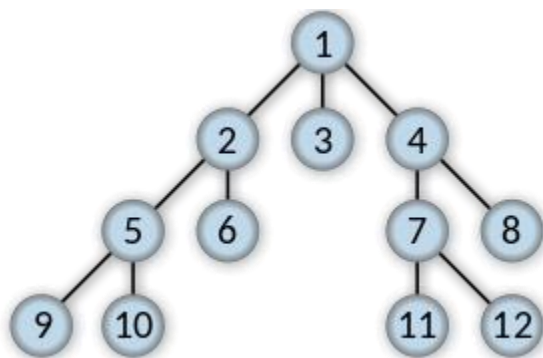


شکل ۳-۴: نظریه گراف [2]

۳-۳- پیمایش در گراف ها و نحوه پیاده سازی آن ها:

پیمایش گراف به معنی بازدید از تک تک رأس های گراف به نحوی خاص است. مسئله پیمایش

درخت حالت خاصی از پیمایش گراف است. برای این چند الگوریتم وجود دارد.



شکل ۳-۵: جستجوی گراف براساس روش BFS [2]

۳-۳-۱- جستجوی اول سطح⁷

الگوریتم جستجوی سطح-اول، گراف را با

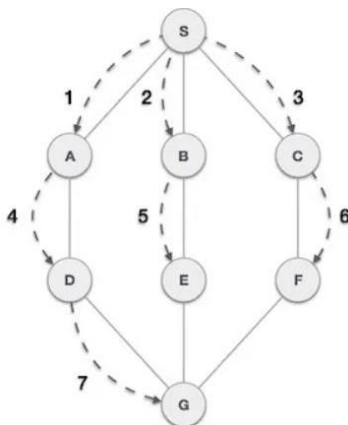
حرکت در سطح پیمایش می کند و از صف برای

به خاطر سپاری

⁷ Breadth_first search

رأس‌های بعدی که می‌خواهد جستجو کند استفاده می‌کند و هر زمان که وارد بن‌بست شود از چرخه

استفاده می‌کند. [2]



شکل ۳-۶: جستجوی گراف براساس BFS. [3]

در مثال فوق الگوریتم سطح-اول ابتدا از رأس A به B به E به F می‌رود و سپس به C و در نهایت از G به D می‌رود. قواعد این پیمایش چنین هستند:

- قاعده اول: از رأس بازدید نشده مجاور بازدید کن. آن را به صورت بازدید شده نشانه‌گذاری کن، آن را نمایش بده و در صف درج کن.
- قاعده ۲: اگر هیچ رأس مجاوری وجود ندارد، نخستین رأس را از صف خارج کن.
- قاعده ۳: قواعد ۱ و ۲ را تا زمانی که صف خالی شود ادامه بده.

پیاده سازی الگوریتم جستجوی سطح اول BFS در زبان C++

```
#include<iostream>
#include <list>

using namespace std;

// This class represents a directed graph using
// adjacency list representation
class Graph
{
    int V;    // No. of vertices
```

```

        // Pointer to an array containing adjacency
        // lists
        list<int> *adj;
public:
    Graph(int V); // Constructor

    // function to add an edge to graph
    void addEdge(int v, int w);

    // prints BFS traversal from a given source s
    void BFS(int s);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}

void Graph::BFS(int s)
{
    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for(int i = 0; i < V; i++)
        visited[i] = false;

    // Create a queue for BFS
    list<int> queue;

    // Mark the current node as visited and enqueue it
    visited[s] = true;
    queue.push_back(s);

    // 'i' will be used to get all adjacent
    // vertices of a vertex
    list<int>::iterator i;

    while(!queue.empty())
    {
        // Dequeue a vertex from queue and print it
        s = queue.front();
        cout << s << " ";
        queue.pop_front();

        // Get all adjacent vertices of the dequeued
        // vertex s. If a adjacent has not been visited,
        // then mark it visited and enqueue it
        for (i = adj[s].begin(); i != adj[s].end(); ++i)
        {
            if (!visited[*i])
            {

```

```

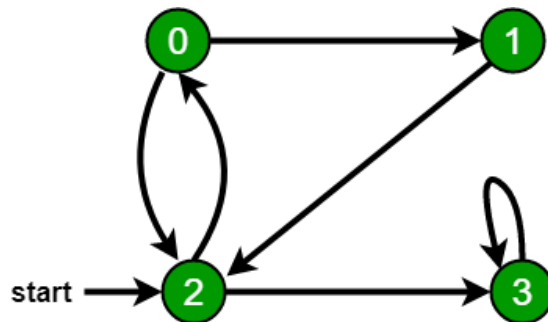
        visited[*i] = true;
        queue.push_back(*i);
    }
}

// Driver program to test methods of graph class
int main()
{
    // Create a graph given in the above diagram
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal "
         << "(starting from vertex 2) \n";
    g.BFS(2);

    return 0;
}

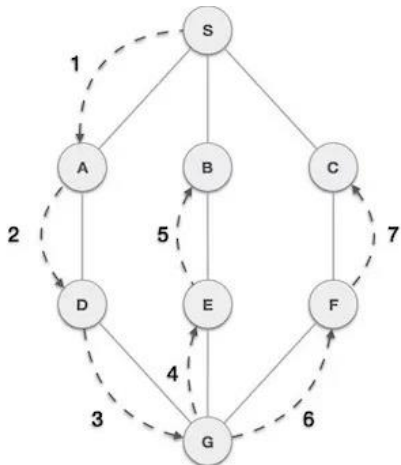
```



شکل ۳-۷: نمونه‌ی عملی از پیمایش گراف براساس الگوریتم BFS [4]

خروجی کد برای گراف بالا براساس الگوریتم پیمایش اول سطح BFS به این صورت است:

Breadth First Search: 2 0 3 1



۳-۲-۳- پیمایش عمق اول^۸

الگوریتم جستجوی عمق -اول طوری یک گراف را پیمایش می‌کند که اولویت آن با پیمایش عمقی گراف است و از یک پشته برای به‌خاطر سپاری رأس‌های بعدی جهت جستجو استفاده می‌کند و زمانی که وارد یک بن‌بست شود، از تکرار بهره می‌گیرد. [3]

شکل ۳-۸: جستجوی گراف براساس DFS. [3]

الگوریتم پیمایش عمق-اول همان طور که در شکل فوق مشخص است، ابتدا از S به A به D به G به E به B می‌رود، سپس به F و در نهایت به C می‌رود. قواعد این پیمایش چنین است:

- قاعده ۱: به رأس بازدید نشده مجاور برو. آن را به صورت بازدید شده علامت‌گذاری کن. آن را وارد پشته بکن.
- قاعده ۲: اگر رأس مجاوری نمانده باشد، یک رأس را از پشته pop کنید. همه رئوسی که رأس‌های مجاور ندارند، از پشته pop می‌شوند.
- قاعده ۳: قاعده ۱ و قاعده ۲ را تا زمانی که پشته خالی شود، ادامه بده.

پیاده سازی الگوریتم جستجوی اول عمق در زبان C++

```
// C++ program to print DFS
// traversal for a given
// graph
#include <bits/stdc++.h>
using namespace std;

class Graph {
```

⁸ Depth First

```

        // A function used by DFS
        void DFSUtil(int v);

public:
    map<int, bool> visited;
    map<int, list<int> > adj;
    // function to add an edge to graph
    void addEdge(int v, int w);

    // prints DFS traversal of the complete graph
    void DFS();
};

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}

void Graph::DFSUtil(int v)
{
    // Mark the current node as visited and print it
    visited[v] = true;
    cout << v << " ";

    // Recur for all the vertices adjacent to this vertex
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFSUtil(*i);
}

// The function to do DFS traversal. It uses recursive
// DFSUtil()
void Graph::DFS()
{
    // Call the recursive helper function to print DFS
    // traversal starting from all vertices one by one
    for (auto i : adj)
        if (visited[i.first] == false)
            DFSUtil(i.first);
}

// Driver Code
int main()
{
    // Create a graph given in the above diagram
    Graph g;
    g.addEdge(0, 1);
    g.addEdge(0, 9);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(9, 3);

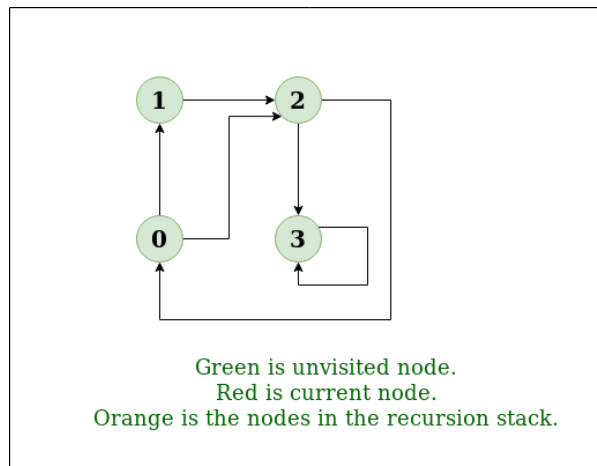
    cout << "Following is Depth First Traversal \n";
    g.DFS();
}

```

```

    return 0;
}

```



شکل ۳-۹: نمونه‌ی عملی از پیمایش گراف براساس DFS [5]

خروجی کد برای گراف بالا براساس جستجوی اول عمق به صورت زیر است:

Output: DFS from vertex 1 : 1 2 0 3

۳-۴- الگوریتم های معروف و نحوه پیاده سازی آن ها

حال که دو نمونه از پرکاربرد ترین الگوریتم های جستجوی گراف ها را دیدیم به سراغ الگوریتم های مشهور حل ماز می رویم و می بینیم به چه صورت در الگوریتم های معروف و صنعتی ماز از الگوریتم های پیمایش گراف ها استفاده شده برای بهینه کردن الگوریتم برای پیدا کردن کوتاه ترین مسیر برای حل ماز.

چند نمونه از الگوریتم های ماز عبارتند از:

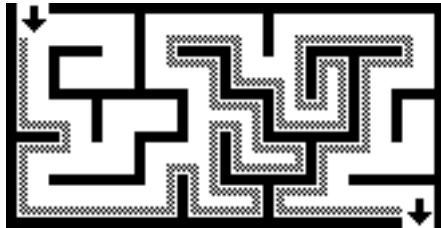
۳-۴-۱- الگوریتم تصادفی ماوس^۹:

این روش یک روش پیش پا افتاده و ساده برای حل ماز است که می تواند توسط یک ربات غیر هوشمند پیاده سازی شود. این روش به این صورت است که ماوس باید مسیر جاری را دنبال کند تا به

^۹ Random mouse algorithm

یک تقاطع برسد و سپس تصمیمی تصادفی در مورد مسیر بعدی باید بگیرد. اگرچه چنین روشی همیشه سرانجام مسیر مناسب را پیدا میکند، اما این الگوریتم به الگوریتم بهینه نیست و بسیار کند است.[6]

۳-۴-۲- الگوریتم دنبال کردن دیواره ها^{۱۰}:

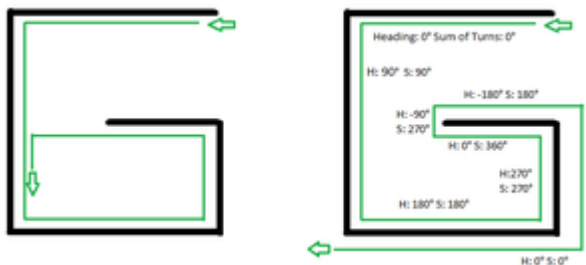


شکل ۳-۱۰: الگوریتم دنبال کردن دیواره ها[4]

یکی از متداول ترین و شناخته شده ترین روش برای عبور از ماز ها با دنبال کردن دیواره ها است. که به عنوان قانون دست راست یا دست چپ نیز شناخته می شود. اگر ماز ساده باشد، یعنی همه ی دیواره ها به یکدیگر یا فضای بیرونی ماز متصل

باشند، این تضمین وجود دارد که با حفظ تماس یک دست به دیواره های که از آن شروع به حرکت می کنیم،

بتوان از Maze خارج شد. ولی Maze ساده نباشد، این الگوریتم به یافتن خروجی در قسمت های گسسته کمکی نمی کند. این الگوریتم یک پیمایش درخت اول عمق است.[6]



شکل ۳-۱۱: الگوریتم PLEDGE . [4]

۳-۴-۳- الگوریتم pledge:

ماز های گسسته را می توان به روش دنبال کردن دیواره ها حل کرد، در صورتی که ورودی و خروجی ماز روی دیواره های خارجی ماز قرار داشته باشند.

¹⁰ Wall follower

چنانچه از درون ماز شروع به حرکت کنیم، ممکن است الگوریتم دنبال کردن دیواره ها در قسمت گسسته ای که شامل خروجی نیست دائما یک حلقه را طی کند. الگوریتم Pledge می تواند این گونه مسائل را حل کند.

الگوریتم Pledge برای رفع موانع، به طی یک مسیر اختیاری نیاز دارد. هنگام مواجهه با مانع، یک دست (مثال دست راست) را در امتداد مانع نگه می داریم در حالیکه زوایای چرخش شمرده می شود. وقتی دوباره در راستای مسیر اصلی قرار گرفتیم و جمع زاویه ای چرخش ها برابر صفر شد، می توان مانع را ترک کرد و در راستای مسیر اصلی حرکت نمود. این الگوریتم به شخص اجازه ی جهت یابی را در شروع از هر نقطه برای خارج از مازهای دوبعدی، را می دهد [6].

۳-۴-۴- Trémaux's الگوریتم

الگوریتم ترمو، که توسط چارلز پیر ترمو^{۱۱} ابداع شد، یک روش کارآمد برای یافتن راه خروج از ماز است که برای مشخص کردن یک مسیر نیاز نیاز به کشیدن خطوط روی زمین دارد و تضمین می شود که برای هر ماز خوش تعریف که تقاطع های کاملاً مشخص دارد کار کند. اما تضمینی برای یافتن کوتاه ترین مسیر وجود ندارد.

الگوریتم براساس قوانین زیر کار می کند:

- هر مسیری را که دنبال میکنید یک بار باید علامت بزنید. علائم باید در هر دو انتهای مسیر قابل مشاهده باشند. بنابراین، اگر آنها به عنوان علائم فیزیکی ساخته می شوند، نه این که به عنوان بخشی از یک الگوریتم رایانه ذخیره شوند، باید همان علامت در هر دو انتهای مسیر ایجاد شود.
- هرگز نباید وارد مسیری که در آن دو علامت وجود دارد شوید

¹¹ Charles Pierre Trémaux

- اگر به تقاطعی رسیدید که هیچ علامتی ندارد (به جز احتمالا موردی که در مسیری که وارد

شده اید)، یک مسیر بدون علامت دلخواه انتخاب کنید، آن را علامت بزنید.

- در غیر این صورت:

- اگر مسیری که وارد آن شده اید فقط یک علامت داشت، بچرخید و از آن مسیر برگردید

و آن را علامت بزنید. به طور خاص باید این مورد هر وقت که بن بست رسیدید رخ دهد.

- اگر نه، یکی از مسیرهای باقیمانده را با کمترین علامت (در صورت امکان بدون علامت،

و در غیر این صورت یک) انتخاب کنید، و آن مسیر را دنبال کنید و آن را علامت بزنید.

قانون بچرخ و برگرد^{۱۲} به طور موثر هر ماز دارای حلقه را به یک ماز ساده تبدیل می کند. هرگاه راهی را

پیدا کنیم که که حلقه ای را ببندد، آن را به عنوان یک بن بست در نظر می گیریم و برمیگردیم. بدون این

قانون یعنی اگر به جای برگشتن به عقب، خودسرانه مسیر دیگری را دنبال کنیم، ممکن است دسترسی به

بخش های هنوز کشف نشده ماز را قطع کنیم. زمانی که در نهایت به هدف رسیدید، مسیرهایی که دقیقا

یکبار علامت گذاری شده اند، درواقع راه بازگشت به نقطه شروع را نشان میدهند. اگر خروجی وجود

نداشته باشد، این روش شما را به نقطه شروع باز میگرداند که در آن همه مسیر ها دوبار علامت گذاری

شده اند. در این مورد، هر مسیر دقیقا دوبار، یک بار در هر جهت طی می شود.

اساسا این الگوریتم که در قرن نوزدهم کشف شد، حدود صد سال بعد به عنوان جستجو در عمق^{۱۳} مورد

استفاده قرار گرفت. که پیش تر راجع به الگوریتم جستجو در عمق در گراف ها صحبت کردیم.[6]

¹² turn around and return rule

¹³ depth-first search.

۳-۴-۵- الگوریتم Dead-end filling

الگوریتم Dead-end filling یک الگوریتم برای حل ماز است که بن بست ها را پر می کند، و تنها راه های صحیح را باز می گذاریم. می توان از این الگوریتم برای حل ماز روی کاغذ یا برنامه های کامپیوتری استفاده کرد. ام این روش برای ماز های نا شناخته مناسب نیست چراکه در این روش باید تمام ماز را بررسی کند. روش بدین صورت است که ابتدا باید همه بن بست ها در ماز را پیدا کنیم و سپس مسیر را از هر بن بست تا رسیدن به اولین تقاطع پر کنیم.[6]

۳-۴-۶- الگوریتم بازگشتی¹⁴

اگر یک پیشنهاد از ماز به شما داده شود، یک الگوریتم بازگشتی ساده می تواند به شما بگوید چگونه به پایان برسد. به الگوریتم یک مقدار X, Y اولیه داده می شود. اگر مقدار X, Y روی دیوار نباشند، متد خود را با تمام مقادیر X, Y مجاور بررسی می کند و مطمئن می شود که قبلا از مقادیر X, Y استفاده نکرده است. اگر مقادیر X, Y مربوط به نقاط انتهایی باشد، تمام نمونه های قبلی را به عنوان مسیر صحیح ذخیره می کند.[6]

این روش درواقع یک پیمایش اول عمق از گراف هاست که بر حسب نقاط سلول ها و یا شبکه بیان می شود.

در ادامه نمونه کدی از الگوریتم بازگشتی در زبان جاوا آمده است:

```
boolean[][] maze = new boolean[width][height]; // The maze
boolean[][] wasHere = new boolean[width][height];
boolean[][] correctPath = new boolean[width][height]; // The solution to
the maze
int startX, startY; // Starting X and Y values of maze
int endX, endY; // Ending X and Y values of maze

public void solveMaze() {
    maze = generateMaze(); // Create Maze (false = path, true = wall)
    for (int row = 0; row < maze.length; row++)
        // Sets boolean Arrays to default values
```

¹⁴ Recursive algorithm

```

        for (int col = 0; col < maze[row].length; col++){
            wasHere[row][col] = false;
            correctPath[row][col] = false;
        }
        boolean b = recursiveSolve(startX, startY);
        // Will leave you with a boolean array (correctPath)
        // with the path indicated by true values.
        // If b is false, there is no solution to the maze
    }
    public boolean recursiveSolve(int x, int y) {
        if (x == endX && y == endY) return true; // If you reached the end
        if (maze[x][y] || wasHere[x][y]) return false;
        // If you are on a wall or already were here
        wasHere[x][y] = true;
        if (x != 0) // Checks if not on left edge
            if (recursiveSolve(x-1, y)) { // Recalls method one to the left
                correctPath[x][y] = true; // Sets that path value to true;
                return true;
            }
        if (x != width - 1) // Checks if not on right edge
            if (recursiveSolve(x+1, y)) { // Recalls method one to the right
                correctPath[x][y] = true;
                return true;
            }
        if (y != 0) // Checks if not on top edge
            if (recursiveSolve(x, y-1)) { // Recalls method one up
                correctPath[x][y] = true;
                return true;
            }
        if (y != height - 1) // Checks if not on bottom edge
            if (recursiveSolve(x, y+1)) { // Recalls method one down
                correctPath[x][y] = true;
                return true;
            }
        return false;
    }
}

```

۳-۴-۷- الگوریتم کوتاه ترین مسیر^{۱۵}

هنگامی که یک ماز چندین راه حل دارد، حل کننده ممکن است بخواهد کوتاه ترین مسیر را از

ابتدا تا انتها پیدا کند. الگوریتم های مختلفی برای پیدا کردن کوتاه ترین مسیرها وجود دارد که بیشتر آن

ها از نظریه گراف ها می آیند که پیشتر راجع به آن ها صحبت کردیم.[6]

¹⁵ Shortest path algorithm

یکی از الگوریتم های کوتاه ترین مسیر را با پیاده سازی یک الگوریتم پیمایش اول-عمق پیدا می کند. در حالی که الگوریتم A^* از تکنیک اکتشافی استفاده می کند. الگوریتم پیمایش سطح اول (BFS) از یک صف برای بازدید از سلول ها به ترتیب فاصله از شروع تا پایان استفاده می کند.

هر سلول بازدید شده باید فاصله خود را از ابتدا دنبال کند یا این که بررسی کند که کدام سلول مجاور نزدیکتر به شروع باعث اضافه شدن آن به صف شده است. هنگامی که مکان انتهایی یافت شد، مسیر سلول ها را به سمت عقب تا شروع کوتاه ترین مسیر دنبال می کند، پیمایش سطح اول در ساده ترین شکل ممکن محدودیت های خود را دارد مانند پیدا کردن کوتاه ترین مسیر در گراف های وزن دار.

۳-۴-۸- الگوریتم Flood Fill

الگوریتم Flood Fill یکی از بهترین الگوریتم های حل ماز است. این الگوریتم مقادیری را به هریک از خانه ها یا سلول ها اختصاص می دهد که این مقادیر نشان دهنده فاصله هر سلول روی ماز تا سلول مقصد است. [7]

طرز کار الگوریتم Flood Fill

طرز کار الگوریتم بر مبنای BFS

۱. یک صف از جفت اعداد ایجاد کنید.
 ۲. مقدار دهی اولیه را وارد صف کنید.
 ۳. یک آرایه دو بعدی برای مکان های بازدید شد ایجاد و مقدار دهی کنید به عنوان مثال $vis[][]$.
 ۴. تا زمانی که صف خالی نباشد مراحل ۴.۱ تا ۴.۶ را تکرار کنید.
- یک عنصر جلویی را از صف بگیرید.
 - آن را از صف پاک کنید.

- ذخیره کردن مقدار/رنگ فعلی در مختصاتی که از صف خارج شده است. (مقدار قبلی)
- مقدار/رنگ فعلی که از صف خارج شده است را بروز کنید.
- هر ۴ جهت را بررسی کنید یعنی $(x,y-1)$ ، $(x,y+1)$ ، $(x-1,y)$ ، $(x+1,y)$ معتبر است یا خیر
- اگر معتبر بود بررسی کنید مقدار آن در آن مختصات باید برابر با مقدار یا رنگ قبلی باشد و مقدار آن در مختصات در آرایه ی `vis[][]` باید ۰ باشد.
- اگر همه شرایط بالا درست باشد، مختصات مربوطه را در صف وارد کنید و به عنوان ۱ در آرایه `vis[][]` قرار دهید.
- آرایه ۲ بعدی را چاپ کنید.
-

پیاده سازی این الگوریتم در زبان C++

```
// C++ program for above approach
#include <bits/stdc++.h>
using namespace std;

// Function to check valid coordinate
int validCoord(int x, int y, int n, int m)
{
    if (x < 0 || y < 0) {
        return 0;
    }
    if (x >= n || y >= m) {
        return 0;
    }
    return 1;
}

// Function to run bfs
void bfs(int n, int m, int data[][8],
         int x, int y, int color)
{
    // Visiting array
    int vis[101][101];

    // Initialing all as zero
    memset(vis, 0, sizeof(vis));

    // Creating queue for bfs
```

```

queue<pair<int, int> > obj;

// Pushing pair of {x, y}
obj.push({ x, y });

// Marking {x, y} as visited
vis[x][y] = 1;

// Until queue is empty
while (obj.empty() != 1)
{
    // Extracting front pair
    pair<int, int> coord = obj.front();
    int x = coord.first;
    int y = coord.second;
    int preColor = data[x][y];

    data[x][y] = color;

    // Popping front pair of queue
    obj.pop();

    // For Upside Pixel or Cell
    if (validCoord(x + 1, y, n, m)
        && vis[x + 1][y] == 0
        && data[x + 1][y] == preColor)
    {
        obj.push({ x + 1, y });
        vis[x + 1][y] = 1;
    }

    // For Downside Pixel or Cell
    if (validCoord(x - 1, y, n, m)
        && vis[x - 1][y] == 0
        && data[x - 1][y] == preColor)
    {
        obj.push({ x - 1, y });
        vis[x - 1][y] = 1;
    }

    // For Right side Pixel or Cell
    if (validCoord(x, y + 1, n, m)
        && vis[x][y + 1] == 0
        && data[x][y + 1] == preColor)
    {
        obj.push({ x, y + 1 });
        vis[x][y + 1] = 1;
    }

    // For Left side Pixel or Cell
    if (validCoord(x, y - 1, n, m)
        && vis[x][y - 1] == 0
        && data[x][y - 1] == preColor)
    {
        obj.push({ x, y - 1 });
        vis[x][y - 1] = 1;
    }
}

```

```

    }
}

// Printing The Changed Matrix Of Pixels
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
    {
        cout << data[i][j] << " ";
    }
    cout << endl;
}
cout << endl;
}

// Driver Code
int main()
{
    int n, m, x, y, color;
    n = 8;
    m = 8;

    int data[8][8] = {
        { 1, 1, 1, 1, 1, 1, 1, 1 },
        { 1, 1, 1, 1, 1, 1, 0, 0 },
        { 1, 0, 0, 1, 1, 0, 1, 1 },
        { 1, 2, 2, 2, 2, 0, 1, 0 },
        { 1, 1, 1, 2, 2, 0, 1, 0 },
        { 1, 1, 1, 2, 2, 2, 2, 0 },
        { 1, 1, 1, 1, 1, 2, 1, 1 },
        { 1, 1, 1, 1, 1, 2, 2, 1 },
    };

    x = 4, y = 4, color = 3;

    // Function Call
    bfs(n, m, data, x, y, color);
    return 0;
}

```

خروجی الگوریتم به صورت زیر در می آید:

```

1 1 1 1 1 1 1 1
1 1 1 1 1 1 0 0
1 0 0 1 1 0 1 1
1 3 3 3 3 0 1 0
1 1 1 3 3 0 1 0
1 1 1 3 3 3 3 0
1 1 1 1 1 3 1 1
1 1 1 1 1 3 3 1

```

خب تا اینجا با معروف ترین الگوریتم های حل ماز آشنا شدیم حالا برای قسمت انتهایی یعنی برای زمانی که ربات اصلی مسیر درست را پیدا کرد و قرار است با ربات همکار جهت تبادل اطلاعات ارتباط برقرار کند صحبت میکنیم که چگونه ربات می تواند با ربات همکار ارتباط برقرار کند.

۳-۵- روش های ارتباطی بین دو دستگاه

روش های مختلفی برای انتقال اطلاعات وجود دارد که هر کدام از آن ها در جای خود کاربردی هستند. می بایست با توجه به کاربرد و امکانات موجود، بهترین روش را انتخاب کرده تا با بهترین هزینه به بالاترین کیفیت دست یابیم.

چون در این پروژه اطلاعات باید به روش بی سیم انتقال یابد بنابراین از روش های دیگر ارتباطی نظیر با سیم صحبت نمی شود.

۳-۶- انتقال بی سیم:

انتقال بی سیم شکلی از رسانه های هدایت نشده است. ارتباط بی سیم هیچ پیوند فیزیکی بین دو یا چند دستگاه وجود ندارد و بصورت بی سیم این ارتباط شکل می گیرد. سیگنال های بی سیم در هوا پخش می شوند و توسط آنتن های مناسب دریافت و تفسیر می شوند. زمانی که آنتن به مدار الکتریکی یک کامپیوتر یا دستگاه بی سیم متصل می شود، داده های دیجیتال را به سیگنال های بی سیم تبدیل می کند و در محدوده فرکانسی خود پخش می شود. گیرنده در سمت دیگر این سیگنال ها را دریافت و آن ها را به داده های دیجیتال تبدیل می کند. یه قسمت کوچکی از طیف های الکترو مغناطیسی را می توان برای انتقال بی سیم استفاده کرد.

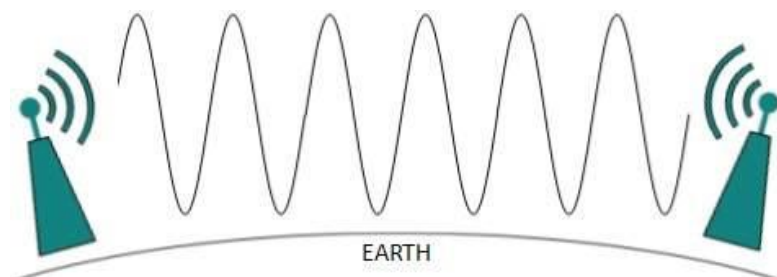


شکل ۳-۱۲: طیف‌های الکترومغناطیسی. [8]

۳-۶-۱- انتقال رادیویی^{۱۶}

فرکانس های رادیویی آسان تر تولید می‌شوند و به دلیل طول موج بالایی که دارند می‌توانند از طریق دیواره ها و ساختارها به طور یکسان نفوذ کند. فرکانس های رادیویی به شش باند تقسیم می‌شوند. امواج رادیویی در فرکانس های پایین تر می‌توانند از دیوارها عبور کنند، درحالی که فرکانس های رادیویی بالاتر می‌توانند در خط مستقیم حرکت کنند و به عقب برگردند. قدرت امواج فرکانس پایین در مسافت های طولانی به شدت کاهش پیدا می‌کند. امواج رادیویی فرکانس بالا قدرت بیشتری دارند.

فرکانس های پایین تر مانند باند های VLF, LF, MF می‌توانند ۱۰۰۰ کیلومتر را بروی سطح زمین



شکل ۳-۱۳: انتقال رادیویی. [8]

طی کنند.

۳-۶-۲- انتقال مایکروویو^{۱۷}

امواج الکترومغناطیسی بالای ۱۰۰ مگاهرتز تمایل دارند در یک خط مستقیم حرکت کنند و

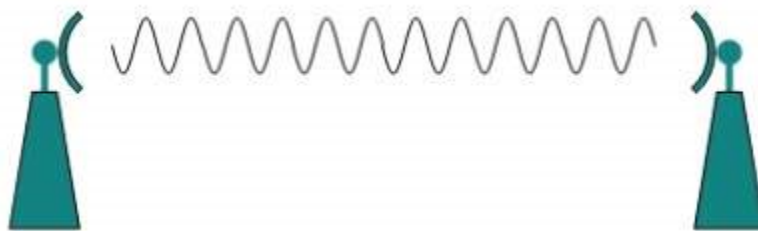
سیگنال ها را می‌توان با تابش آن امواج به سمت یک ایستگاه خاص ارسال کرد. از آنجایی که امواج

¹⁶ Radio Transmission

¹⁷ Microwave Transmission

مایکروویو در خطوط مستقیم حرکت می‌کنند، فرستنده و گیرنده هردو باید در یک راستا باشند تا کاملاً در خط دید هم دیگر قرار بگیرند.

امواج مایکروویو می‌توانند دارای طول موج بین ۱ میلی‌متر تا ۱ متر و فرکانس بین ۳۰۰ مگاهرتز تا ۳۰۰ گیگاهرتز باشند.



شکل ۳-۱۴: انتقال مایکروویو. [8]

آنتن‌های مایکروویو امواج را متمرکز می‌کنند و پرتویی را از آن می‌سازند. همانطور که در تصویر بالا نشان داده شده است، چندین آنتن را می‌توان برای رسیدن به نقاط دورتر تراز کرد. امواج مایکروویو فرکانس بالاتری دارند و در دیوار نفوذ نمی‌کنند.

انتقال مایکروویو به شدت به شرایط آب و هوایی و فرکانس مورد استفاده بستگی دارد.

۳-۶-۳- انتقال مادون قرمز^{۱۸}

موج مادون قرمز بین طیف مرئی و امواج مایکروویو قرار دارد. دارای طول موج ۷۰۰ نانومتر تا ۱ میلی‌متر و محدوده فرکانس از ۳۰۰ گیگاهرتز تا ۴۰۰ تتراهرتز است.

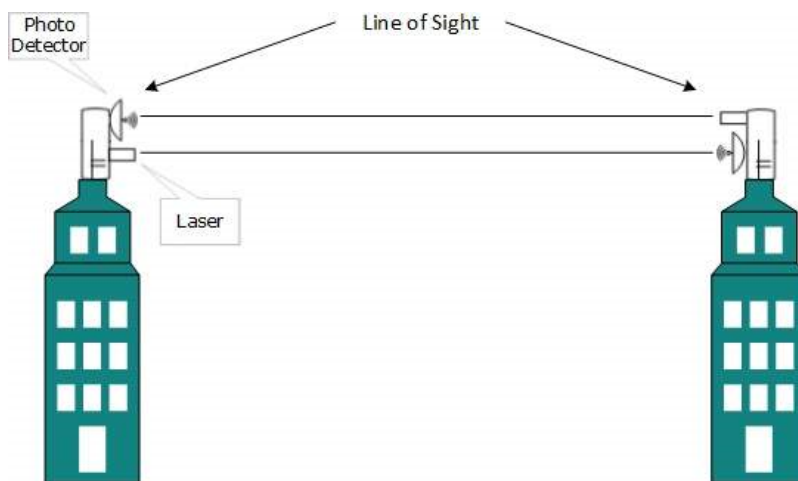
امواج مادون قرمز برای اهداف ارتباطی با برد بسیار کوتاه همانند تلویزیون و ریموت‌ها استفاده می‌شود. مادون قرمز در یک خط مستقیم حرکت می‌کند، ازین رو طبیعتاً جهت‌دار است. به دلیل محدوده فرکانس بالا، مادون قرمز نمی‌تواند از موانع مانند دیوار عبور کند.

¹⁸ Infrared Transmission

۳-۶-۴- انتقال نوری^{۱۹}

بالاترین طیف الکترومغناطیسی که می‌تواند برای انتقال داده استفاده شود، سیگنالینگ اپتیک یا نوری است. که این امر به وسیله لیزر محقق می‌شود.

به دلیل استفاده از نور، فرکانس تمایل دارد که کاملاً در خط مستقیم حرکت کند. بنابراین فرستنده و گیرنده باید در خط دید هم دیگر باشند. از آنجایی که انتقال لیزر یک جهته است، در هر دو انتهای ارتباط، لیزر و ردیاب باید عکس هم نصب شوند. پرتو لیزر عموماً ۱ میلی متر عرض دارد، در نتیجه دقت کار برای تراز کردن دو گیرنده که دور از هم قرار دارند به وسیله منابع لیزر بالاست.



شکل ۳-۱۵: انتقال نوری [8]

لیزر به عنوان فرستنده (TX) و آشکارسازهای عکس به عنوان گیرنده (RX) کار می‌کند.

لیزرها نمی‌توانند از موانعی مانند دیوارها، باران، مه غلیظ نفوذ کنند. علاوه بر آن، پرتو لیزر توسط باد، دمای اتمسفر یا تغییر دما در مسیر تغییر می‌کند. لیزر یک انتقال داده امن است زیرا آسیب زدن به لیزر با عرض ۱ میلی متر بدون قطع کانال ارتباطی بسیار دشوار است.

¹⁹ Light Transmission

خب راجع به روش های انتقال داده بی سیم به صورت کلی صحبت شد. به طور خاص برای این پروژه از روش وایرلس برای ایجاد این ارتباط بین دو دستگاه استفاده شده است .

برای ارتباط وایرلس بین دو دستگاه از ماژول NRF24L01 استفاده شده است، راجع این ماژول و روش راه اندازی آن به طور مفصل در فصل بعد پرداخته شده است.



شکل ۳-۱۶: ماژول وایرلس NRF24L01

این چیپ در واقع یک ماژول بسیار عالی برای ارسال و دریافت اطلاعات از طریق وایرلس با خطای بسیار پایین است. چون فرکانس بالایی در حدود ۲.۴ گیگاهرتز دارد براحتی از اجسامی همانند دیوار و موانع دیگر عبور می کند و باعث می شود برد بیشتری به ما بدهد.

فصل چهارم

ابزارها و راه‌اندازی آن‌ها

۴-۱- معرفی میکروکنترلر^{۲۰} استفاده شده

ابتدا به تعریفی از میکروکنترلر داشته باشیم ببینیم میکروکنترلر چیست؟

میکروکنترلر چیست؟

میکروکنترلر (به انگلیسی: Microcontroller) گونه‌ای ریزپردازنده است که دارای حافظه دسترسی تصادفی (RAM) و حافظه فقط‌خواندنی (ROM)، تایمر، پورت‌های ورودی و خروجی (I/O) و درگاه ترتیبی (Serial Port) پورت سریال) درون خود تراشه است، و می‌تواند به تنهایی ابزارهای دیگر را کنترل کند. به عبارت دیگر یک میکروکنترلر، مدار مجتمع کوچکی است که از یک CPU کوچک و اجزای دیگری مانند تایمر، درگاه‌های ورودی و خروجی آنالوگ و دیجیتال و حافظه تشکیل شده‌است. میکروکنترلر یا ریز کنترل گر یک تراشه IC است که درواقع مغز ربات شما است که برای کنترل دستگاه‌های دیگر برنامه نویسی می‌شود.

انواع میکروکنترلر

میکروکنترلرها انواع گوناگونی دارند. میکروکنترلرهای ARM ، میکروکنترلرهای AVR ، میکروکنترلر atx mega، میکروکنترلرهای plc ، میکروکنترلرهای ۸۰۵۱ و غیره بخشی از انواع میکروکنترلر است. برای ساخت ربات‌های میکروماوس بسته به وقت و هزینه و کارایی که از ربات انتظار دارید می‌توانید یکی از میکروکنترلرها را انتخاب کنید.

خب با توجه به وقت محدودی که در اختیار داشتیم از پلت فرم آردوینو^{۲۱} استفاده کردیم که این پلت فرم از میکرو کنترلر AVR Atmega328 ساخته شده است.

حال برای آن که بیشتر با آردوینو آشنا بشیم به تصویر زیر نگاه کنید که یک نمونه از پلت فرم آردوینو است.

²⁰ Microcontroller

²¹ Arduino

آردوینو

آردوینو (به انگلیسی: Arduino) یک پلتفرم سخت‌افزاری و نرم‌افزاری متن‌باز است. پلتفرم آردوینو شامل یک میکروکنترلر تک‌بردی متن‌باز است که قسمت سخت‌افزار آردوینو را تشکیل می‌دهد. علاوه بر این، پلتفرم آردوینو یک نرم‌افزار آردوینو IDE که به منظور برنامه‌نویسی برای بردهای آردوینو طراحی شده است و یک بوت لودر نرم‌افزاری که بر روی میکروکنترلر بارگذاری می‌شود را در بر می‌گیرد. پلتفرم آردوینو به منظور تولید سریع و ساده پروژه‌های سخت‌افزاری تعاملی و ساخت وسایلی که با محیط تعامل داشته باشند طراحی شده است، البته بردهای آردوینو اهداف آموزشی را نیز دنبال می‌کنند.



شکل ۴-۱: آردوینو مدل اونو

توجه داشته باشید همانطور که در توضیحات بالا اشاره شد آردوینو ابزار مناسبی برای ساخت دستگاه‌های صنعتی و حرفه‌ای نیست و صرفاً به عنوان یک ابزار مناسب و سریع و ارزان برای اهداف آزمایشگاهی و آموزشی و پروژه‌های ساده و نیمه حرفه‌ای دانشگاهی و ... مناسب است. اگر قصد ساخت ربات حرفه‌ای برای مقاصد تجاری و صنعتی و مسابقات مختلف رسمی را دارید آردوینو گزینه مناسبی نیست.

۴-۲- انتخاب سنسور ها و نحوه راه‌اندازی آن

خب بعد از این که میکرو مورد نظر را انتخاب کردیم برای این که بتوانیم مانع‌ها را در مسیر تشخیص دهیم باید سنسور هایی را انتخاب کرد که بتوانند این کار را انجام دهند که در ادامه به چند نمونه از این سنسور ها اشاره شده است.

سنسور:

سنسور (sensor) یعنی حس کننده و از کلمه sens به معنی حس کردن گرفته شده و می تواند کمیت هایی مانند فشار، حرارت، رطوبت، دما، و ... را به کمیت های الکتریکی پیوسته (آنالوگ) یا غیرپیوسته (دیجیتال) تبدیل کند.

اما سنسورها در ربات :

سنسورها اغلب برای درک اطلاعات تماسی، تنشی، مجاورتی، بینایی و صوتی به کار میروند. عملکرد سنسورها بدینگونه است که با توجه به تغییرات فاکتوری که نسبت به آن حساس هستند، سطوح ولتاژی ناچیزی را در پاسخ ایجاد میکنند، که با پردازش این سیگنالهای الکتریکی میتوان اطلاعات دریافتی را تفسیر کرده و برای تصمیمگیریهای بعدی از آنها استفاده کرد.

خب در این پروژه چون با سنسورهای محیطی سرکار دارد فقط به این سنسورها اشاره شده است.

سنسور محیطی چیست؟

این سنسورها درواقع اطلاعات رو از محیط خارج و موقعیت ربات با اشیای اطراف را دریافت می کنند. برای این که بتوان موقعیت ربات با موانع اطرافش را دریافت کرد می توان از سنسور هایی نظیر سنسور نوری، سنسور حسگر مجاورتی مادون قرمز، آلتراسونیک، سنسور IR و.. استفاده کرد.

سنسور شدت نور

برخی از سنسورها تنها دو پایه دارند. این سنسورها درواقع، مقاومت

هایی هستند که در اثر قرار گیریدر وضعیت های مختلف، مقاومت آن ها

تغییر می کند.

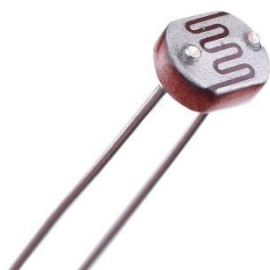
سنسور شدت نور یکی از این سنسورهاست. در اثر افزایش شدت نور،

مقاومت الکتریکی این سنسور کاهش می یابد.

با توجه به این که آردوینو نمی تواند مقاومت را مستقیما اندازه گیری کند،

باید به طریقی تغییرات در مقاومت سنسور را به تغییرات ولتاژ تبدیل کرد.

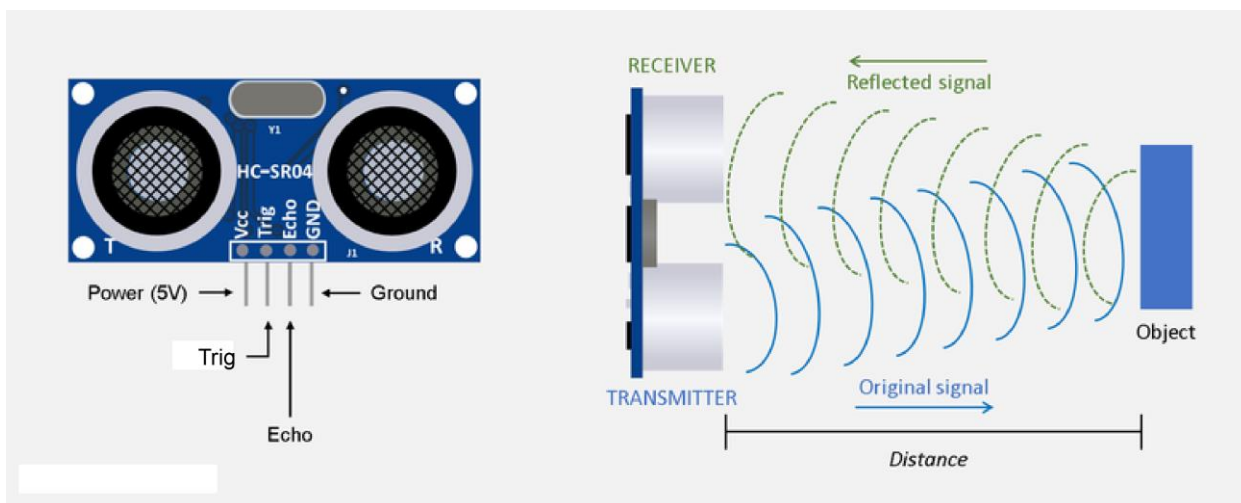
یکی از رایج ترین روش ها برای این کار، استفاده از تقسیم ولتاژ است.



شکل ۴-۲: سنسور شدت نور

۳-۴- سنسور فاصله سنج اولتراسونیک^{۲۲}

ماژول فاصله سنج اولتراسونیک با ارسال یک موج صوتی و محاسبه زمان بازگشت موج به سنسور پس از برخورد با مانع، فاصله سنسور تا مانع را محاسبه کند. این سنسور برای ساخت ربات های ماز و ربات های obstacle avoidance و کاربردهایی ازین قبیل استفاده می شود.



شکل ۳-۴: طرز کار ماژول اولتراسونیک

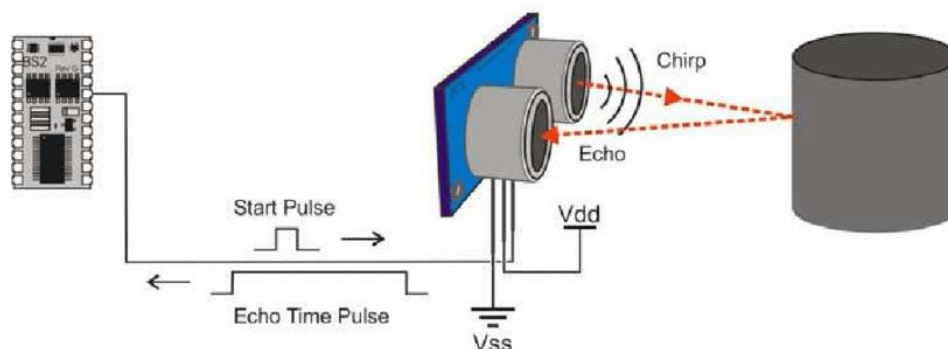
همانطور که در تصویر بالا می بینید ماژول HC-SR04 چهار پایه دارد. پایه های VCC و GND به ترتیب به ۵ ولت و GND آردوینو متصل می شود. و دو پایه دیگر یعنی Trig و Echo به پین های دیجیتال آردوینو متصل می شود هرچند در آردوینو میتواند پایه های آنالوگ را هم به دیجیتال تبدیل کرد بنابراین در صورت کمبود فضا می توانید از پایه های آنالوگ آردوینو به عنوان پایه های دیجیتال استفاده کنید

۳-۴-۱- طرز کار اولتراسونیک:

با ارسال موج فراصوتی توسط سنسور فاصله سنج اولتراسونیک و محاسبه زمان بازگشت آن پس از برخورد با مانع، میتوان فاصله سنسور تا مانع را بدست آورد. امواج فراصوتی مانند امواج صوتی، با سرعت ۳۴۰ متر بر ثانیه در فضا حرکت می کنند.

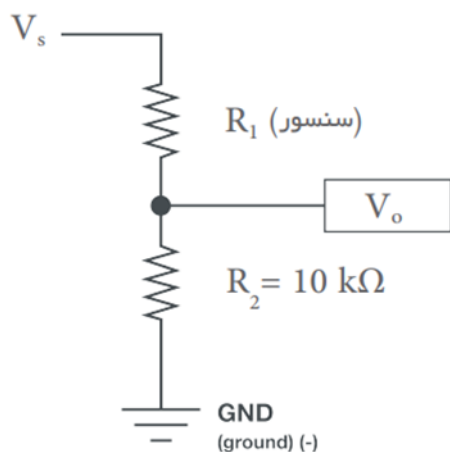
²² Ultrasonic sensor

لازم به ذکر است که مسافت طی شده توسط موج فراصوتی از ارسال تا برگشت، دو برابر فاصله سنسور از مانع است؛ در نتیجه برای بدست آوردن فاصله، باید مسافت طی شده توسط موج، در بازه ارسال تا دریافت موج را تقسیم بر دو کرد.



شکل ۴-۴: طرز کار ماژول اوولتراسونیک

باتوجه به سرعت بالای موج فراصوتی، در محاسبات و کدنویسی از واحد میکروثانیه (10^{-6} ثانیه) استفاده می‌شود. در نتیجه سرعت صوت به جای ۳۴۰ متر بر ثانیه ۰.۰۳۴ سانتی متر بر میکروثانیه محاسبه می‌شود.



$$V_o = V_s * \frac{R_2}{R_1 + R_2} \quad (1)$$

شکل ۴-۵: مدار راه اندازی سنسور شدت نور

۴-۳-۲- نحوه ایجاد موج فراصوتی:

برای ایجاد موج فراصوتی، ابتدا ۲ میکروثانیه، پایه Trig در وضعیت LOW قرار داده می‌شود تا اطمینان حاصل شود که پین Trig از قبل در وضعیت HIGH قرار ندارد. سپس به مدت ۱۰ میکروثانیه،

این پین در وضعیت HIGH قرار داده می‌شود. با استفاده از دستور pulseIn در آردوینو، زمان رسیدن پالس ارسالی به پین Echo اندازه‌گیری و به مسافت تبدیل می‌شود.

۳-۳-۴ نحوه عملکرد pulseIn

دستور pulseIn منتظر می‌ماند که pin، در وضعیت فعال قرار گیرد به عنوان مثال

```
pulseIn(echoPin,HIGH);
```

در دستور بالا آردوینو منتظر می‌ماند که پین Echo در وضعیت HIGH قرار گیرد وقتی یک پالس HIGH به پین می‌رسد، دستور pulseIn، مدت زمانی که پین در وضعیت LOW بوده تا ابتدای دریافت پالس HIGH را برمی‌گرداند.

۴-۳-۴ کد نوشته شده برای اندازه‌گیری مسافت سنسور تا مانع در آردوینو:

```
int trigPin=3;
int echoPin=2;
long duration;
int distance;
int incomingByte = 0; // for incoming serial data

void setup() {
  pinMode(trigPin,OUTPUT);
  pinMode(echoPin,INPUT);
  Serial.begin(9600);
}

void loop() {
  Serial.print("Distance: ");
  Serial.println(getDistanceFromSensor());
}

int getDistanceFromSensor() {
```



```

digitalWrite(trigPin,LOW);
delayMicroseconds(2);
digitalWrite(trigPin,HIGH);
delayMicroseconds(10);
digitalWrite(trigPin,LOW);
duration=pulseIn(echoPin,HIGH);
distance=duration*0.034/2;
return distance;
}

```

در این پروژه از سه سنسور اولتراسونیک برای تشخیص موانع در جهات مختلف استفاده شده. در صورت وقت کافی می‌توانید از ترکیب سنسور های نوری و IR و اولتراسونیک استفاده کنید برای بالا بردن دقت هرچند باید به سائز ربات خود هم دقت کنید بعد سنسور مورد نظر را انتخاب کنید.

استفاده از سه سنسور برای اینکار اشکالاتی را ایجاد کرد و این حالت استفاده از سه سنسور اولتراسونیک برای ساخت ربات های حرفه ای اصلا توصیه نمی‌شود و این که با وجود سه سنسور اما باز هم قسمت هایی از فضا را ممکن است پوشش ندهد. اگر قرار است از این سنسور ها در بهینه ترین حالت استفاده کنید بهتر است از یک سروو موتور ۱۸۰ درجه استفاده کنید که در این حالت ما فقط نیاز به یک سنسور اولتراسونیک داریم و چرخش های ۱۸۰ درجه سروو موتور باعث می‌شود فضای بسیار بیشتری توسط سنسور پوشش داده شود و همچنین دقت کار بالا رود، اما همیشه دقت، کاهش سرعت را به همراه دارد و با اینکار عملا سرعت را از دست می‌دهید ولی در عوض دقت را بدست می‌آورید این حالت برای ربات هایی که نیاز به سرعت دارند مانند ربات های میکرو ماوس و .. مناسب نیست و برای اینکار باید سنسور های مناسب به تعداد مورد نیاز برای پوشش کل فضا و همچنین دقت کافی استفاده شود.

۴-۴- انتخاب موتور

موتورها یکی از مکانیزم های اصلی حرمت ربات ها هستند. موتور ها انواع مختلفی دارند که هر کدام از آنها وظایف خاصی را انجام می دهند. بعضی از موتور ها را می توان به چرخ هایی متصل کرد که روبات را به اطراف حرکت دهد. بر اساس پروژه مورد نظر باید موتور خود را انتخاب کنیم که برای اینکار باید یکسری فاکتورها در انتخاب موتور را در نظر بگیریم که در ادامه به آن اشاره می کنیم.

عوامل مهم در انتخاب موتور :

۱. سرعت : سرعت همان حداکثر دور موتور است که معمولاً با دور در دقیقه یا RPM اندازه گیری می شود. ۱ دور در دقیقه به این معنی است که محور موتور هر دقیقه یک بار کاملاً به دور دایره می چرخد که بسیار کند است. حتی یک موتور DC بسیار ارزان نیز دارای سرعت حداقل ۱۰۰۰ RPM است.

۲. گشتاور : گشتاور اندازه گیری میزان قدرت خروجی موتور است. یعنی حداکثر وزنی که موتور می تواند بلند کند چقدر است.

به طور معمول ، هرچه دور موتور بیشتر باشد ، گشتاور آن کمتر است و بالعکس. بنابراین در طراحی و ساخت ربات، پیدا کردن موتورهایی با تعادل مناسب سرعت و گشتاور امری بسیار مهم است.

محرک های ربات

سه نوع محرک در ربات ها وجود دارد:

- Pneumatic actuators
- Hydraulic actuators
- Electric actuators

اکثریت قریب به اتفاق ربات ها از موتورهای الکتریکی استفاده می کنند که اغلب از موتورهای DC گیربکس دار و بدون جاروبک در روبات های قابل حمل یا موتورهای AC در ربات های صنعتی و ماشین های CNC استفاده می کنند. این موتورها اغلب در سیستم هایی با بارهای سبک تر ترجیح داده می شوند و شکل غالب حرکت چرخشی است. [8]

که در اینجا به طور خاص روی موتور های الکتریکی تمرکز شده است. موتورهای الکتریکی: این نوع موتور ها انرژی الکتریکی را به انرژی مکانیکی تبدیل می کنند. امروزه این نوع موتور ها رایج ترین نوع موتور در ساخت ربات ها است. آن ها همچنین در ربات هایی با اندازه های مختلف به خوبی عمل می کنند، نسبت به توان خروجی خود فشرده هستند و دارای آلودگی ناچیز هستند. اگرچه موتور های الکتریکی رایج ترین نوع موتور در ساخت ربات ها هستند اما مشکلاتی نیز دارند نظیر این که سیستم فرمان پیچیده تری در مقایسه با سایر انواع موتورها نیاز دارند.

انواع موتور های الکتریکی

- موتورهای جریان متناوب ²³(AC): جریان متناوب در پریزهای دیواری رایج موجود است. موتورهای AC از این منبع انرژی برای تولید القای الکترومغناطیسی استفاده می کنند. مهندسان از مکانیسم های AC در موقعیت هایی که نیاز به سرعت ثابت دارند، استفاده می کنند. با این حال، این موتورها برای استفاده در صنعت رباتیک مناسب نیست و برای مقیاس های صنعتی بزرگ و با گشتاور بالا هستند.

²³ Alternating current (AC) motors

- موتور های جریان مستقیم ^{۲۴}(DC): معمولا این نوع موتور ها با باتری تغذیه می شوند. مکانیسم های DC در طیف وسیعی از اندازه ها ظاهر می شوند و محدوده های بار بسیار متغیر، به علاوه زمان پاسخ و تحرک سریع در مقایسه با مدل هایی که از برق شهر تغذیه می شوند ارائه می کنند.
- سروو موتور ^{۲۵} ها: این موتور ها به طور خلاصه “Servo” نامیده می شوند، از جمله سروو های صنعتی، برخی از دقیق ترین موتور های موجود هستند. آنها در حین کار از تنظیمات خطای فوری پشتیبانی می کنند. اگر به موقعیت یابی دقیق به همراه گشتاور بالا در پکیج کوچک نیاز دارید، این موتور ها برای بازوهای رباتیک و دیگر انواع ربات و کوبات ^{۲۶} عالی هستند. هنگام کار با سروو ها ممکن است بین حرکات تاخیر وجود داشته باشد.
- موتور های پله ای ^{۲۷} یا استپر موتور ها : استپر موتور یا استپ موتور یا موتور پله ای یک موتور براشلس الکتریکی dc است که یک دور کامل ۳۶۰ درجه را به تعدادی پله یا استپ مساوی تقسیم میکند. این موتور ها می توانند تا هنگام نگه داشتن یک جسم، گشتاور بالا و همچنین موقعیت یابی دقیق برای هر مرحله را فراهم کنند. دقت این موتور ها در حدود ۰.۰۱ درجه و ۰.۱ میلی متر است. گشتاور کلی مشابه موتور های سروو است. کارکرد موتور های پله ای نسبت به انواع دیگر از نظر تامین انرژی الکتریکی مورد نیاز تا حدودی گران تر است.

چگونه موتور مناسب برای ربات خود انتخاب کنیم؟

برای انتخاب موتور برای پروژه های رباتیک بعدی خود می توانید نکات بالا را مرور کنید. با این حال قبل این که یکبار دیگر موارد بالا را بخوانید، لیستی از ویژگی هایی که در ربات خود نیاز دارید ایجاد کنید.

²⁴ Direct current (DC) motors

²⁵ Servo motors

²⁶ cobot

²⁷ Stepper motors

میتوانید به عنوان مثال این سوال ها را در لیست خود بپرسید.

ربات چه وظیفه‌ای را انجام خواهد داد، و میزان خطای شما برای حرکات آن چقدر باید باشد؟

محیط مورد نظر را از نزدیک مطالعه کنید تا درک لازم برای آلودگی نیز درک کنید، آلودگی موتور نه تنها عامل اصلی خرابی موتورها است، بلکه تهدیدی برای کیفیت محصول در محیط های تولیدی ریسک گریز هست.

نکات اصلی دیگر در انتخاب موتور که باید به آن دقت شود:

- نسبت اندازه به توان:^{۲۸} در چه مقیاسی می‌سازید؟ آیا ربات نیاز به حرکت در محیط های باریک دارد یا قدرت موتور مهم تر از اندازه است؟
- محدودیت های بار:^{۲۹} انواع درایوهایی که برای ربات خود انتخاب می‌کنید به گشتاور کافی برای مقابله با بارهای مورد انتظاری که حمل می‌کند نیاز دارند.
- دقت: وقتی ربات در حال حرکت هست، تحمل انحراف چقدر است؟ آیا به موتوری نیاز دارید که تصحیح خطا را در لحظه انجام دهد؟
- قابلیت اطمینان:^{۳۰} بخش تعمیر و نگهداری شما چقدر پرسنل دارد؟ هرچند وقت یکبار میخواهید تعمیر و نگهداری ربات خود را انجام دهید؟
- مصرف منابع: موتور های پله ای در حالت آماده به کار نیز انرژی مصرف می‌کنند، اما گاهی اوقات آنها تنها انتخاب هستند.

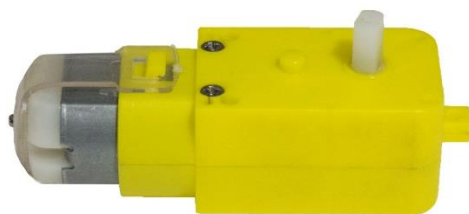
²⁸ Size to power ratio

²⁹ Load limits

³⁰ Reliability

خب راجع موتور های مختلف صحبت شد که بسته به پروژه مورد نظر باید انتخاب شوند چرا که بدون موتور ها عملاً ربات هیچ است و باید در انتخاب موتور دقت لازم را بشود.

اما برای این پروژه چون وقت بسیار محدود بود و باید سرعت تصمیم گیری میشد از موتور های گیربکسی پلاستیکی استفاده شده است.



شکل ۴-۶: موتور گیربکسی پلاستیکی

همانطور که در تصویر بالا می بینید موتور های گیربکسی پلاستیکی به این شکل هستند، اما متأسفانه این نوع موتور ها که در بازار هست از کیفیت بسیار پایینی برخوردار هستند و تراز کردن موتور ها برای این که با سرعت مشابه حرکت کنند کار دشواری است. چرا که این موتور ها رو برد به خوبی عمل نمی کنند و در پیاده سازی رباتان به درستی عمل نمی کنند و انتظارات را برآورده نمی کند در این پروژه برای هر ربات از دو عدد از این موتور های گیربکسی استفاده شده اما روی برد متأسفانه یکی از موتور جریان بیشتری

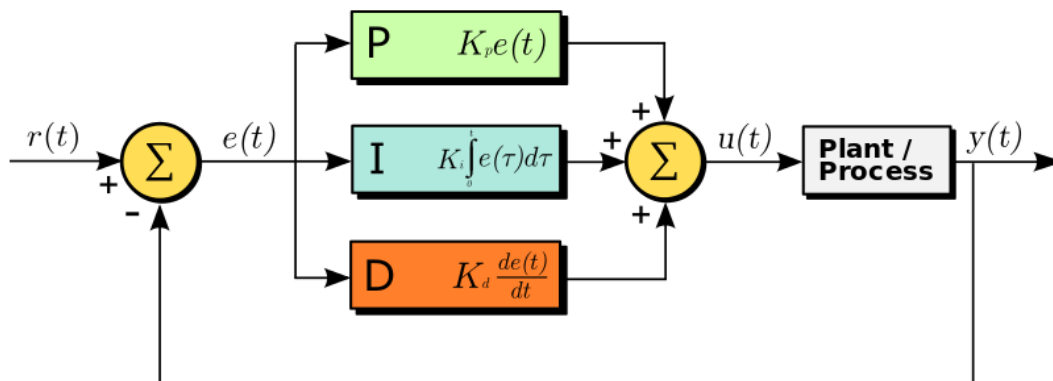
می کشد و تند تر می چرخد. که برای حل این موضوع می توان از ترانزیستور های قدرتی نظیر TIP42 به همراه خازن عدسی استفاده شود تا جریان و ولتاژ تقریباً یکسان به موتور ها برسد و موتور های تان بصورت مشابه حرکت کنند. اما این موتور ها از دقت خوبی برخوردار نیستند و بعد از تجربه کردن پیشنهاد می شود از این نوع موتور ها برای ساخت چنین ربات هایی استفاده نشود تنها مزیت این نوع موتور ها قیمت آن است که ارزان قیمت هستند اما برای مصارف حرفه ای و دقیق مناسب نیستند. در صورت داشتن بودجه کافی از موتور های با کیفیت تر DC یا استپر موتور ها یا سروو موتور ها با قابلیت تنظیم PWM استفاده کنید تا در اجرا پروژه به مشکل نخورید و ربات بدرستی تست ها را پشت سر بگذارد.

همچنین یکی دیگر از راهکارهایی که می‌توان برای حل این موضوع استفاده کرد، استفاده از PID کنترلر است. که در سیستم های کنترل خطی با این کنترل کننده آشنا شدیم.

۴-۵- PID کنترلر چیست؟

PID مخفف انتگرال^{۳۱} و مشتق^{۳۲} است. این نام از روش هایی در مورد نحوه برخورد چنین کنترل کننده ای با اغتشاش در سیستم گرفته شده است. با این حال، چنین کنترل کننده ای فقط در سیستم های دارای فیدبک وجود دارد.

سیستم فیدبک دار سیستمی است که بخشی از خروجی به ورودی فیدبک می‌شود.



شکل ۴-۷: یک سیستم فیدبک با کنترل کننده PID

عملکرد این کنترل کننده به این صورت است که سیگنال یا سیگنال هایی را به عنوان خطا از ورودی دریافت می‌کند و سپس عملیاتی را روی آن انجام می‌دهد و در نهایت خروجی شان باهم جمع می‌شود. خروجی این مجموعه همان خروجی کنترل کننده PID است که برای اصلاح خطا به سیستم فیدبک داده می‌شود.

فرمول استاندارد PID به شکل زیر است:

³¹ Integral

³² Derivative

$$y(t) = K_p (e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T d \frac{de}{dt}) \quad (2)$$

با توجه به رابطه بالا تابع تبدیل به صورت زیر به دست می آید:

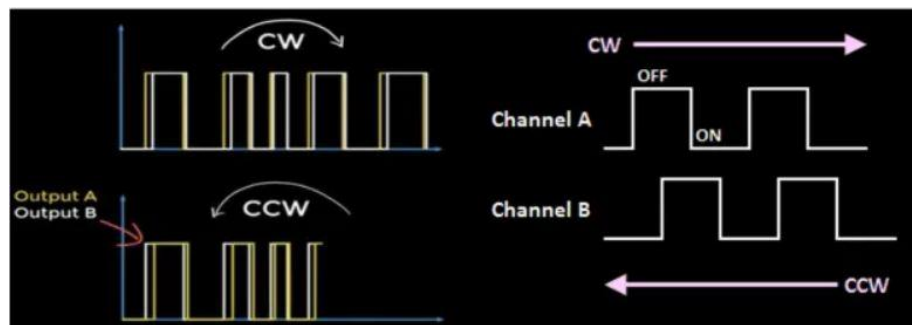
$$G_c = K_p + \frac{K_i}{s} + K_d s \quad (3)$$

با استفاده از PID کنترلر می توان خطا را تا حد زیادی کاهش داد و سرعت موتور ها با تقریب زیادی مشابه شوند.

۴-۵-۱- پیاده سازی PID کنترلر با استفاده از Arduino :

برای اینکه بتوان از PID کنترلر استفاده شود باید از موتور های دارای انکودر چرخشی^{۳۳} یا رمزگذار چرخشی استفاده شود تا با استفاده از آن بتوان موقعیت شفت و در نهایت موقعیت چرخ های ربات را مشخص کرد.

کاری که باید انجام داد این است که هر زمان که چرخ از موقعیت خود خارج شد، موتور بچرخد، علاوه بر این موتور از طریق مدولاسیون عرض پالس، یا همان PWM کنترل می شود هرچه عرض پالس بیشتر باشد موتور تندتر می چرخد.



شکل ۴-۸: کنترل موتور ها از طریق PWM با PID کنترلر

³³ rotary encoder

۴-۵-۲- کد نوشته شده در نرم افزار Arduino برای پیاده سازی PID کنترلر:

```
#include <PID_v1.h>

#define PIN_INPUT 0
#define PIN_OUTPUT 3

//Define Variables we'll be connecting to
double Setpoint, Input, Output;

//Specify the links and initial tuning parameters
double Kp=2, Ki=5, Kd=1;
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);

void setup()
{
    //initialize the variables we're linked to
    Input = analogRead(PIN_INPUT);
    Setpoint = 100;

    //turn the PID on
    myPID.SetMode(AUTOMATIC);
}

void loop()
{
    Input = analogRead(PIN_INPUT);
    myPID.Compute();
    analogWrite(PIN_OUTPUT, Output);
}
```

۴-۶- درایور موتور

یک کنترل کننده موتور یک دستگاه الکترونیکی است (معمولا به شکل یه مدار بدون پوشش و محفظه است) که به عنوان یک دستگاه واسطه بین میکروکنترلر، یک منبع تغذیه یا باتری و موتورهای عمل می‌کند. اگرچه میکروکنترلر (مغز ربات) سرعت و جهت موتور را مشخص میکند، اما به دلیل محدودیت زیاد در تغذیه خروجی (جریان و ولتاژ) نمی‌تواند آن‌ها را مستقیما هدایت کند. از طرف دیگر درایور موتور میتواند جریان را در ولتاژ مورد نظر فراهم کند اما نمیتواند تصمیم بگیرد که موتور تا چه میزان سریع بچرخد. بنابراین، میکروکنترلر و کنترل کننده موتور باید باهم کار کنند تا موتور به طور مناسبی حرکت کند. معمولا میکروکنترلر میتواند از طریق یک روش ارتباطی ساده مانند UART(serial) یا PWM به کنترل کننده موتور دستورالعملی برای چگونگی تغذیه موتور بدهد. هم چنین برخی از کنترل کننده های موتور را می‌توان به صورت دستی و با استفاده از ولتاژ آنالوگ کنترل کرد (معمولا با یک پتانسیومتر ایجاد می‌شود). اندازه و وزن فیزیکی کنترل کننده موتور می‌تواند بسیار متفاوت باشد، از یک دستگاه کوچکتر از نوک انگشتان برای کنترل یک ربات کوچک sumo تا یک کنترل کننده سنگین وزن چند کیلوگرمی. اندازه و وزن کنترل کننده ربات معمولا کمینه تاثیر را روی ربات دارد، تا زمانی که شما با ربات‌های خیلی کوچک و یا هواپیماهای بدون سرنشین کار می‌کنید که در این حالت کوچکترین وزن‌ها هم تاثیرگذار خواهد بود. اندازه کنترل کننده موتور معمولا به حداکثر جریانی که میتواند فراهم کند وابسته است. جریان بیشتر به معنی استفاده از سیم‌هایی با قطر بزرگتر است.

درایور موتور Motor Driver از مدارهای مهم در رباتیک و پروژه‌های مختلف می‌باشد. دسته بندی درایور

موتور بر اساس نوع تراشه و میزان ولتاژ و به خصوص جریان عبوری از درایور اهمیت دارد. انواع درایور

موتور Motor Driver برای فرمان به موتور DC موتور Stepper و یا سرو موتور Servo Motor Driver

را بررسی کنید. از آنجایی که راه اندازی موتورهای مختلف نیاز به تامین جریان و ولتاژ مجزا دارد، برای

مدیریت موتور Motor هم نیاز به تراشه و مدار کنترل موتور **Motor Driver** می‌باشد. همانطور که

توضیح داده شد، بر اساس جریان عبوری و ولتاژ قابل عبور توسط ماژول دسته بندی می‌شوند.

درایور موتور Motor Driver یا راه انداز موتور برای به حرکت در آوردن موتور به کار می‌روند. این نوع

مدارها به ازای اطلاعات دریافتی از میکرو و یا کنترل کننده، ولتاژ مورد نیاز موتور را تامین می‌کنند.

۴-۶-۱- درایورهای مختلف موتور برای ربات های خود ران:

همانطور که پیش‌تر اشاره شد ربات های خودران توسط یک میکروکنترلر کنترل می‌شوند. از

درایور های Fireblade و Viper Motor می‌توان برای ارتباط میکرو با موتور استفاده کرد. به غیر از این

درایو ها چند درایور موتور با جریان پایین نظیر L298N ، L9110 ، L293D و... وجود دارد.

L293D در شیلدهای مختلف مانند Auton Shield, Xbee Shield, Starter Shield استفاده می‌شوند.

چراکه در موتورهای جریان پایین که معمولا در ربات های خودران کوچک استفاده می‌شود بهترین است.

L298N یک درایور دو موتوره است که می‌تواند به شما کمک کند تا موتور هایی که حداکثر ۲ آمپر

جریان دارند مانند موتور های BO ، موتور های گیربکسی متصل کند. برای موتور های رابطی که جریان

بالا می‌کشند، می‌توان سری Fireblade 30A , Fireblade 10A – Fireblade series (بدون کنترل

سرعت) یا درایور موتور وایپر^{۳۴} (کنترل کننده سرعت) با توجه به کاربرد مورد نظر انتخاب کرد. در این

پروژه به طور خاص دو ماژول درایور L298N و L9110 راه اندازی می‌شود.

³⁴ Viper motor driver

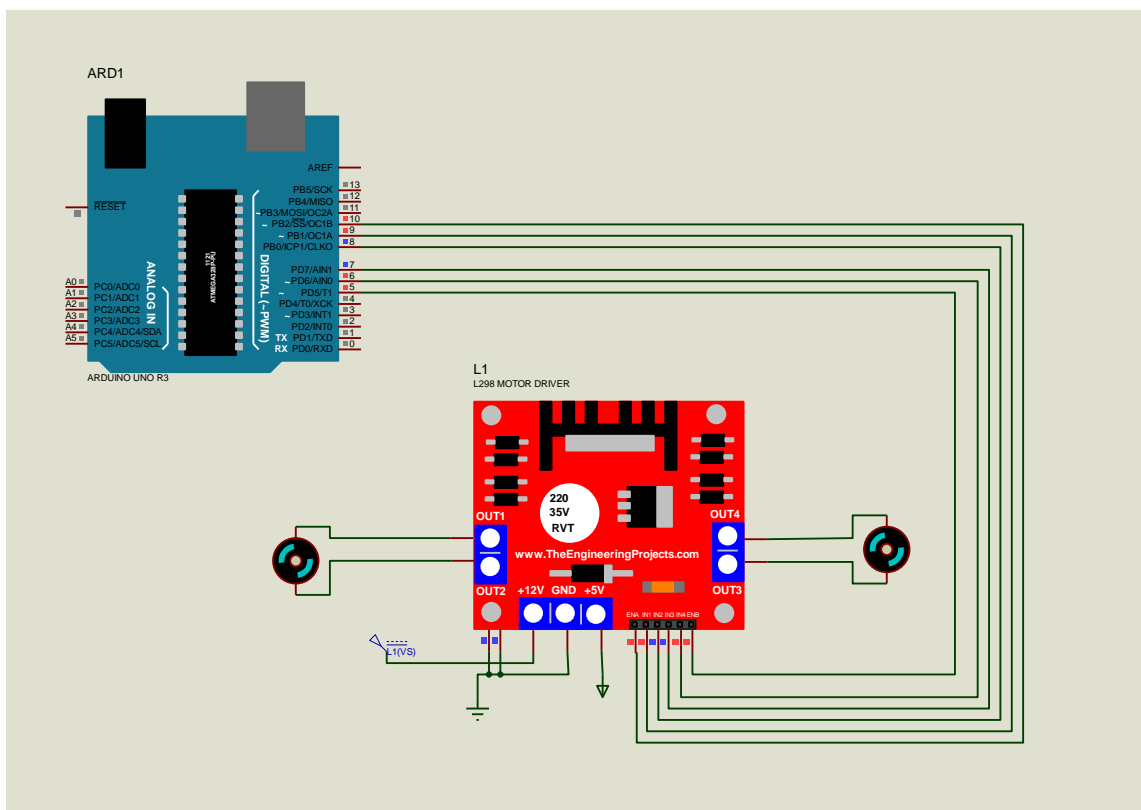
۷-۴- ماژول درایور L298N و راه اندازی آن :



این درایور یکی از ارزان ترین و کاربردی ترین ماژول های کنترل موتور در بازار می باشد که می توان با آن دو موتور dc و بعضی از استپ موتور ها را با این درایور راه اندازی کرد.

شکل ۴-۹: ماژول درایور L298N

۷-۴-۱- راه اندازی درایور L298N با آردوینو



شکل ۴-۱۰: مدار راه انداز موتورهای dc به وسیله ماژول Lm298N و آردوینو در نرم افزار پروتئوس

همانطور که در تصویر بالا مشاهده می شود برای راه اندازی موتور های dc با استفاده از این ماژول باید به این صورت عمل شود. موتور ها باید به پایه های Out1 و Out2 و یا Out3 و Out4 متصل شوند. پایه ۱۲ ولت ماژول باید به سر مثبت باتری متصل شود و پایه Gnd باید به سر منفی باتری و آردوینو و پایه ۵ ولت ماژول به ۵ ولت آردوینو متصل شود.

سپس پایه های ورودی مازول به پایه های دیجیتال آردوینو متصل شود و اینکه برای کنترل سرعت موتور باید پایه های En مازول به پایه های pwm آردوینو که با علامت `~` مشخص شده است متصل شود تا به توان از طریق دستور `analogWrite(en, pwm)` آردوینو می توان سرعت موتور ها را کنترل کرد.

۴-۷-۲- کد نوشته شده برای راه اندازی مازول Lm298N

کد نوشته شده براساس مدار بالا که در شکل (۴-۱۰) نشان داده شده است نوشته شده است:

```
#define MA1 6 // Motor A pins
#define MA2 7
#define MB1 8 // Motor B pins
#define MB2 9
#define EnB 10
#define EnA 5
#define PWM_RIGHT 200
#define PWM_LEFT 200

void setup() {
    // put your setup code here, to run once:
    pinMode(MA1, OUTPUT);
    pinMode(MA2, OUTPUT);
    pinMode(MB1, OUTPUT);
    pinMode(MB2, OUTPUT);
    pinMode(EnA, OUTPUT);
    pinMode(EnB, OUTPUT);
}

void loop() {
    forward();
    delay(1000);
    Stop();
}

void forward() { //function of forward
```

```

digitalWrite(MA1, HIGH);
digitalWrite(MA2, LOW);
digitalWrite(MB1, LOW);
digitalWrite(MB2, HIGH);
analogWrite(EnA, PWM_LEFT);
analogWrite(EnB, PWM_RIGHT);
}

void backward() {           //function of backward
    digitalWrite (MA1, HIGH);
    digitalWrite (MA2, 0);
    digitalWrite (MB1, HIGH);
    digitalWrite(MB2, 0);
    analogWrite(EnA, PWM_LEFT);
    analogWrite(EnB, PWM_RIGHT);
}

void Stop() {               //function of stop
    digitalWrite(MA1, LOW);
    digitalWrite(MA2, LOW);
    digitalWrite(MB1, LOW);
    digitalWrite(MB2, LOW);
    analogWrite(EnA, PWM_LEFT);
    analogWrite(EnB, PWM_RIGHT);
}

void turnLeft() {          //function of turn left
    digitalWrite(MA1, HIGH);
    digitalWrite(MA2, LOW);
    digitalWrite(MB1, LOW);
    digitalWrite(MB2, HIGH);
    analogWrite(EnA, PWM_LEFT);
    analogWrite(EnB, PWM_RIGHT);
}

```

```

}

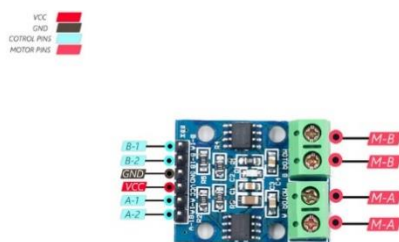
void turnRight() {           //function of turn right
    digitalWrite(MA1, LOW);
    digitalWrite(MA2, HIGH);
    digitalWrite(MB1, HIGH);
    digitalWrite(MB2, LOW);
    analogWrite(EnA, PWM_LEFT);
    analogWrite(EnB, PWM_RIGHT);
}

```

۸-۴- مازول درایور L9110S و راه اندازی آن :

این مازول یک مازول درایور دو کاناله جمع وجور و ارزان قیمت برای راه اندازی موتور ربات های کوچک است. این مازول دارای دو تراشه درایور موتور مستقل است که هر کدام از آن ها می توانند تا ۸۰۰ میلی آمپر جریان را افزایش دهند. ولتاژ کاری این مازول بین ۲.۵ تا ۱۲ ولت است، که این مازول را قادر می سازد با هر میکروکنترلر ۳.۳ و ۵ ولت استفاده شود.

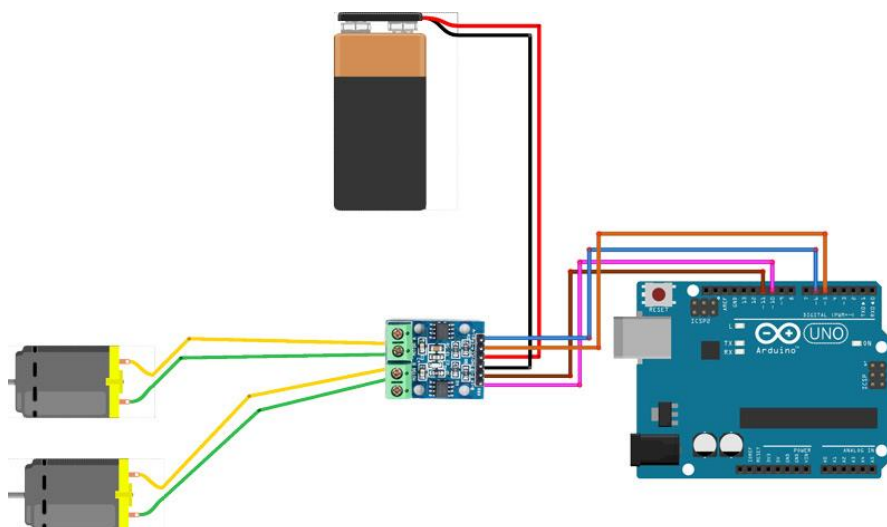
موتورها از طریق دو ترمینال پیچی به مازول متصل می شوند. سیگنال مدولاسیون عرض پالس PWM برای کنترل سرعت موتور و یک خروجی دیجیتال برای تغییر جهت استفاده می شود. مازول همچنین می تواند برای راه اندازی استپر موتورهای ۴ سیمه ۲ فاز نیز استفاده شود. نصب آن روی ربات یا پروژه های دیگر به آسانی انجام می شود.



شکل ۴-۱۱: مازول درایور L9110S

۴-۸-۱- راه اندازی ماژول درایور L9110S با آردوینو:

راه اندازی این ماژول درایو بسیار شبیه به ماژول L298N است با این تفاوت که این ماژول پایه En ندارد.



شکل ۴-۱۲: مدار راه انداز ۲ موتور DC به وسیله درایور L9110S با آردوینو

۴-۸-۲- کد نوشته شده برای راه اندازی ماژول L9110S

کد نوشته شده برای مدار بالا که در شکل (۴-۱۲) نشان داده شده است نوشته شده است:

```
#define MA1 8 // Motor A pins
#define MA2 9
#define MB1 10 // Motor B pins
#define MB2 11
#define PWM_RIGHT 200
#define PWM_LEFT 200

void setup() {
    pinMode(MA1, OUTPUT);
    pinMode(MA2, OUTPUT);
    pinMode(MB1, OUTPUT);
    pinMode(MB2, OUTPUT);
}

void loop() {
    forward();
}
```



```

    delay(1000);
    turnLeft();
    delay(1000);
    turnRight();
    delay(1000);
    Stop();
}

void forward() {           //function of forward
    analogWrite(MA1, PWM_RIGHT);
    digitalWrite(MA2, LOW);
    digitalWrite(MB1, LOW);
    analogWrite(MB2, PWM_Left);
}

void backward() {          //function of backward
    analogWrite(MA1, PWM_RIGHT);
    digitalWrite (MA2, 0);
    analogWrite(MB1, PWM_RIGHT);
    digitalWrite(MB2, 0);
}

void Stop() {              //function of stop
    digitalWrite(MA1, LOW);
    digitalWrite(MA2, LOW);
    digitalWrite(MB1, LOW);
    digitalWrite(MB2, LOW);
}

void turnLeft() {          //function of turn left
    analogWrite(MA1, PWM_RIGHT);
    digitalWrite(MA2, LOW);
    digitalWrite(MB1, LOW);
    analogWrite(MB2, PWM_LEFT);
}

void turnRight() {         //function of turn right

```

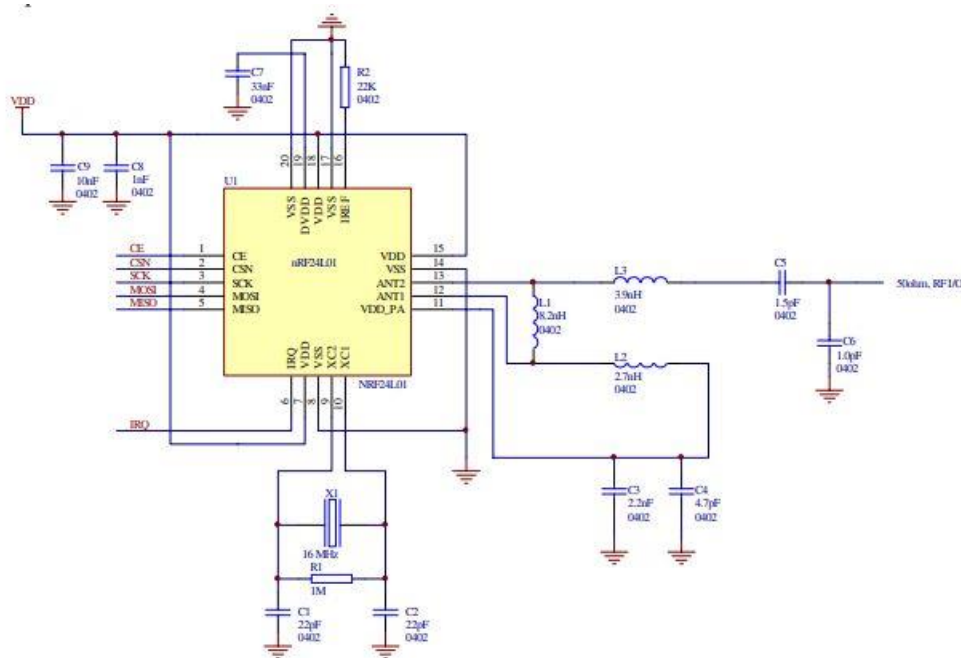
```
digitalWrite(MA1, LOW);
analogWrite(MA2, PWM_RIGHT);
analogWrite(MB1, PWM_LEFT);
digitalWrite(MB2, LOW);
}
```

۴-۹- مازول وایرلس nRF24L01 و راه اندازی آن

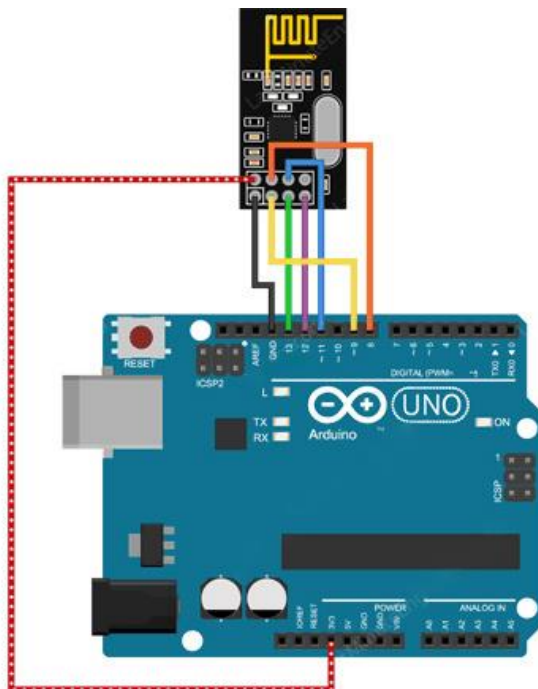
ارتباط بیسیم دو یا چند برد آردوینو در بسیاری از کاربردهای صنعتی نظیر مانیتور کردن داده سنسورها، کنترل ربات‌ها، اتوماسیون خانگی و ... مورد نیاز خواهد بود. یک راه حل ارزان قیمت و مطمئن برای ایجاد ارتباط RF دو طرفه استفاده از مازول nRF24L01+ که یک مازول فرستنده-گیرنده تولید شرکت Nordic Semiconductor است. nRF24L01 یک فرستنده رادیویی تک تراشه‌ای برای فرکانس کاری 2.4-2.5GHz در باند ISM است. فرستنده و گیرنده از یک سینت سائزر^{۳۵} فرکانسی کاملاً یکپارچه، یک منبع تغذیه، یک تقویت کننده، یک نوسان ساز کریستالی، یک دمدولاتور، مدلاتور و موتور پروتکل توان خروجی پیشرفته ShockBurstTM، کانال های فرکانسی تشکیل شده است. و به راحتی قابل برنامه ریزی از طریق رابط SPI با مصرف جریان بسیار پایین است.[12]

³⁵ synthesizer

۴-۹-۱- راه اندازی ماژول nRF24L01:



شکل ۴-۱۳: شماتیک nRF24L01 برای خروجی ۵۰ اهم RF [12]



شکل ۴-۱۴: نحوه راه اندازی ماژول nRF24L01 با Arduino

۴-۹-۲- کد نوشته شده برای راه اندازی ماژول nRF24L01

الگوریتم نوشته شده براساس مدار بالا که در شکل (۴-۱۴) نشان داده شده است نوشته شده است

کد آردواینو فرستنده

```
//Include Libraries
#include < SPI.h >
#include < nRF24L01.h >
#include < RF24.h >
//create an RF24 object
RF24 radio(9, 8); // CE, CSN
//address through which two modules communicate.
const byte address[6] = "00001";
void setup()
{
    radio.begin();
    //set the address
    radio.openWritingPipe(address);
    //Set module as transmitter
    radio.stopListening();
}
void loop()
{
    //Send message to receiver
    const char path[] = "Correct Path";
    radio.write(&path, sizeof(path));
    delay(1000);
}
```

کد آردواینو گیرنده

```
//Include Libraries
#include < SPI.h >
#include < nRF24L01.h >
#include < RF24.h >

//create an RF24 object
RF24 radio(9, 8); // CE, CSN

//address through which two modules communicate.
const byte address[6] = "00001";

void setup()
{
    while (!Serial);
    Serial.begin(9600);

    radio.begin();

    //set the address
    radio.openReadingPipe(0, address);

    //Set module as receiver
    radio.startListening();
}

void loop()
{
}
```

```
//Read the data if available in buffer
if (radio.text style="color:blue;">available())
{
  char path[32] = {0};
  radio.read(&path, sizeof(path));
  Serial.println(path);
}
}
```

۴-۱۰- انتخاب منبع تغذیه

منبع تغذیه ربات های کوچک را باتری ها تشکیل می دهند. باتری ها انواع مختلفی دارند که باید براساس پروژه موزد نظر انتخاب شوند. یکپاز نکاتی که باید در انتخاب باتری ها دقت شود، قابل شارژ بودن یا نبودن آن است. برای صرفه جویی در هزینه انتخاب باتری قابل شارژ می تواند بهترین گزینه باشد، درست است در ابتدا هزینه بیشتری پرداخت خواهید کرد اما با چند بار شارژ کردن این باتری هزینه ها جبران خواهد شد. یکی دیگر از مواردی که اهمیت دارد، اندازه است. به طور کلی باتری ها با سلول های بزرگ تر عمر طولانی تری دارند. ولتاژ های بالاتر به سلول های بیشتری نیاز دارد. در حالی که ممکن است پردازنده شما احتمالا فقط به منبع تغذیه ۵ ولت در چند ۱۰ میلی آمپر نیاز داشته باشد درحالی که موتور ها و سنسور ها به ولتاژ های بالا و ظرفیت زیاد نیاز دارند.

به عنوان مثال یک ربات میکروماوس با موتور پله ای^{۳۶} برای اینکه بتواند بهترین عملکرد دینامیکی خود را داشته باشد، نیاز به یک یا چند جفت سلول برای دادن ولتاژ ۱۵ یا بیشتر داشته باشد، هنگامی که استپر ها کاملا انرژی دارند. ممکن است ۲ آمپر بکشند ربات شما باید حداقل ۱۵ دقیقه در حالت آماده به کار بماند. که در این شرایط مدیریت منابع تغذیه مهم خواهد شد. موتورها باید در صورت عدم نیاز خاموش شوند، سنسور ها باید برای مدت زمان کمتری پالس شوند و از بیش از حد چشمک زدن چراغ ها جلوگیری شود. این یک پیکره بندی آسان برای مدیریت بهتر منبع تغذیه بود. به عنوان مثال افت ۸ ولت

³⁶ Stopper Motor

در ۲۰۰ میلی آمپر رگولاتورها را بسیار داغ می کند که برای جلوگیری از داغ شدن رگولاتورها باید از هیت سینک استفاده شود.

۴-۱۱- پارامترهای عملکرد باتری

ولتاژ پایانه :

ولتاژ بین پایانه های باتری را ولتاژ پایانه می نامند که با ولت اندازه گیری می شود. معمولاً یک سلول، ولتاژی از ۱ تا ۲ ولت دارد (اگر دانشمندان از گاز فلورین با پتانسیل کاهش ۲.۸۷ و فلز لیتیوم با -۳.۰۵ برای گرفتن ۵.۹۲ ولت استفاده کنند، چه می شود؟ فقط یک خیال پردازی ...). با اتصال چندین باتری (در واقع سلول) به صورت سری، ولتاژ بالاتری را می توان به دست آورد.

ولتاژ مدار باز :

وقتی باتری نه در حال شارژ است و نه خالی شدن، در این هنگام ولتاژ پایانه به عنوان ولتاژ مدار باز شناخته می شود.

منحنی ولتاژ :

منحنی ولتاژ یک باتری افت ولتاژ پیشرونده را هنگام تخلیه نشان می دهد.

منحنی تخلیه :

باتری ها معمولاً تمایل دارند هنگام کار از لحاظ ولتاژ افت کنند. تعداد کمی از باتری ها ولتاژ اولیه خود را تا زمان تخلیه کامل حفظ می کنند. این تخلیه در ولتاژ به صورت نموداری در برابر زمان نشان داده می شود. هر چه منحنی صاف تر باشد، باتری بهتر است؛ اکثر باتری های قدیمی دارای تخلیه شیب دار بودند و باتری های جدیدتر البته صاحب برند معتبر، منحنی تخلیه صاف دارند. خیلی بعید است بتوانید برای یک

باتری قلمی عادی که از بازار می خرید، خصوصاً در ایران انتظار دریافت منحنی باتری داشته باشید. چرا که اغلب فروشندگان قطعات تخصصی متاسفانه و با ابراز ناراحتی زیاد از این مساله، سواد و تخصص کافی در ارائه اطلاعات فنی محصول خود را ندارند! غیر از این مورد تولید کننده هم ممکن است برای یک باتری با کاربرد ساده مثلاً اسباب بازی و مشابه آن منحنی ارائه ندهد. اما در لوازم حساس تر مثلاً یک ربات پرنده، یک دستگاه نظامی خرید باتری خیلی شوخی به نظر نمی رسد.

ظرفیت ذخیره سازی :

مقدار جریانی است که باتری می تواند در واحد زمان تأمین کند که با آمپر-ساعت اندازه گیری می شود (برای باتری ها معمولاً میلی آمپر ساعت است). مثلاً اگر باتری ۲۰۰۰ میلی آمپر ساعت است، یعنی باتری می تواند ۲ آمپر یا ۲۰۰۰ میلی آمپر جریان را برای مدت یک ساعت تأمین کند. اگر ربات فقط ۱۰۰۰ میلی آمپر جریان مصرف می کند، باتری حدوداً ۲ ساعت کار می کند. حالا دیگر باید تفاوت بین mA و mAh این است که در مقایسه باتری ها با یکدیگر آمپر ساعت از آمپر تنها خیلی مهم تر است. معمولاً روی تمام باتری ها عدد میلی آمپر نوشته شده. با اتصال موازی باتری ها (در واقع سلول ها) می توان جریان خروجی بالاتر را بدست آورد.

: C-rate

این مورد برای یک طراح ربات خیلی مهم نیست، اما خُب C-rate ... میزان شارژ و دشارژ باتری است که با توجه به ظرفیت ذخیره سازی آن که معمولاً mAh یا Ah است، بیان می شود. ۱C به معنای تخلیه کل انرژی ذخیره شده در ۱ ساعت است و ۰.۵C به معنای تخلیه کل انرژی در ۲ ساعت است. مثلاً، یک باتری با ۱.۵ میلی آمپر ساعت، در صورت تخلیه با ۱C، جریان ۱۵۰۰ میلی آمپر، برای یک ساعت می دهد. اگر همان باتری ۰.۵C باشد، ۷۵۰ میلی آمپر در ۲ ساعت تخلیه می شود. معمولاً اکثر باتری ها ۱C هستند.

توان :

مقدار توان باتری در واحد حجم است که با وات بر متر مکعب اندازه گیری می شود .

تعداد دوره :

به تعداد دفعاتی گفته می شود که باتری شارژ و خالی (قابل استفاده برای باتری های شارژشونده) می شود، پیش از اینکه عملکرد به زیر حد انتظار برسد .

ماندگاری :

مدت زمانی که باتری می تواند بدون استفاده در قفسه یا فروشگاه سالم بماند .

طول عمر :

مدت زمان تا پیش از افت عملکرد باتری؛ چه استفاده شده چه استفاده نشده .

دما :

عملکرد اکثر باتری ها با تغییر دما کاهش می یابد. بهتر است که باتری انتخاب کنید که با کمی تغییر دما عملکرد آن افت نکند .

انواع باتری ها

- لیتیم
- نیکل - کادمیم

- باتریهای هیبرید نیکل - فلز³⁷

- باتریهای لیتیم - یون³⁸

- ...

برای اینکه ربات عملکرد بهتری داشته باشد و موتور ها و سنسور ها در بهترین عملکرد خود قرار داشته باشند بهتر است منبع تغذیه میکروکنترلر را از سنسورها و موتور ها مستقل شود.

در این پروژه برای میکرو از باتری کتابی ۹V و منبع تغذیه موتور ها و سنسور ها نیز از باتری لیتیوم یونی ۱۸۶۵۰ استفاده شده است.

از مزایای باتری لیتیوم یونی میتوان به موارد زیر اشاره کرد:

- باتری های لیتیوم یونی عمر طولانی دارند

- قابل شارژ می باشند

- چگالی انرژی بالایی دارند

- و ...

در کنار مزیت هایی که دارند معایبی نیز دارا هستند از جمله این که این باتری ها در مقایسه با سایر باتری ها:

- از وزن بالایی برخوردارند

- فضای زیادی را اشغال می کنند

- و همچنین تخلیه شارژ نسبتا بالایی دارند.

³⁷ nickel-metal hybrid

³⁸ Lithium-Ion

خب تا اینجا با وسایلی که می‌توان یک ربات میکروماوس و ربات حل ماز هوشمند با ویژگی هایی که در پروژه تعریف شده ساخت و نحوه راه اندازی آن ها صحبت شد.

در بخش بعدی راجع به پیاده سازی الگوریتم و پیاده سازی نهایی ربات صحبت خواهد شد.

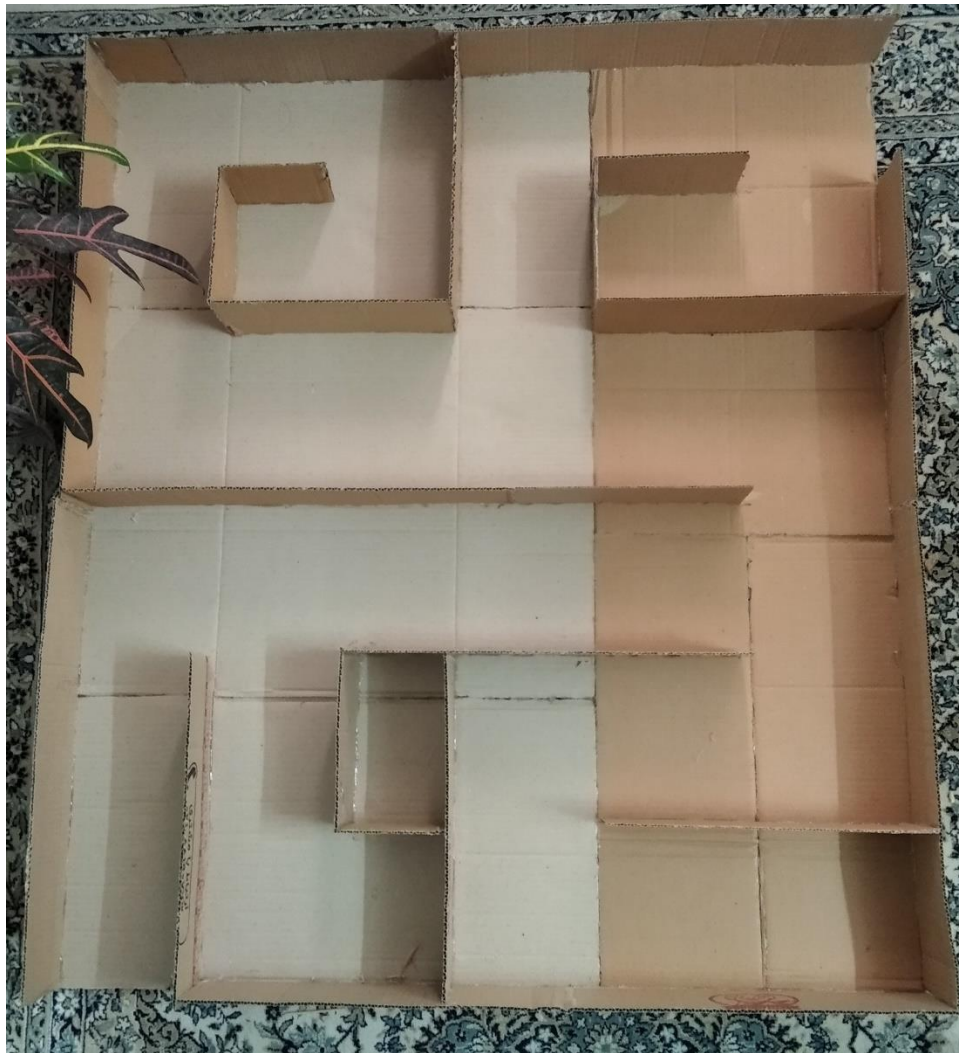
فصل پنجم

پیاده سازی در عمل

۵-۱- ساخت مسیر

خب در بخش های قبل راجع به کلیات پروژه و شیوه انجام آن به صورت تئوری صحبت شد، حالا در این قسمت یعنی بخش آخر راجع پیاده سازی عملی مطالب گفته شده در بخش های قبل پرداخته می شود.

خب حالا باید ماز طراحی شده را واقعیت پیاده سازی شود. شکل ۵.۱ پیاده سازی ماز انتخابی در محیط واقعی می باشد.



شکل ۵-۱: پیاده سازی ماز طراحی شده

۵-۲- ساخت ربات

در قسمت های قبل راجع به ابزارهایی که می شود یک ربات میکروماوس یا ربات حل ماز را ساخت

صحبت شد.

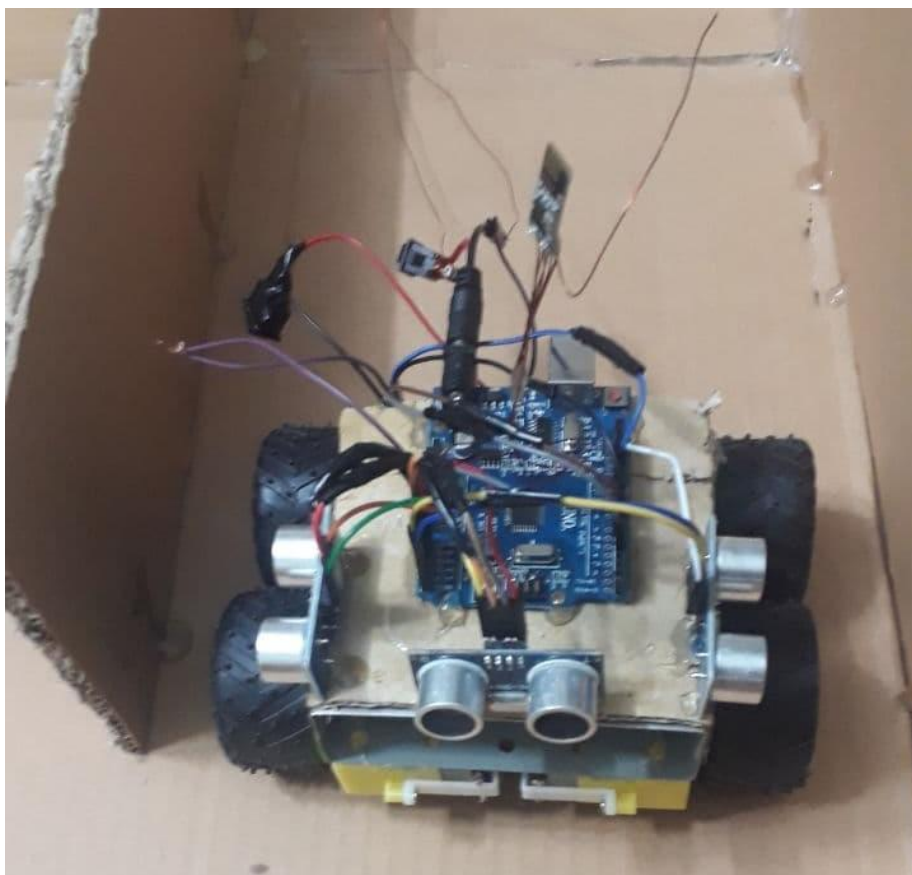
در این قسمت به صورت عملی ربات حل مسئله ماز برای این پروژه طراحی و ساخته می شود.

وسایل مورد نیاز:

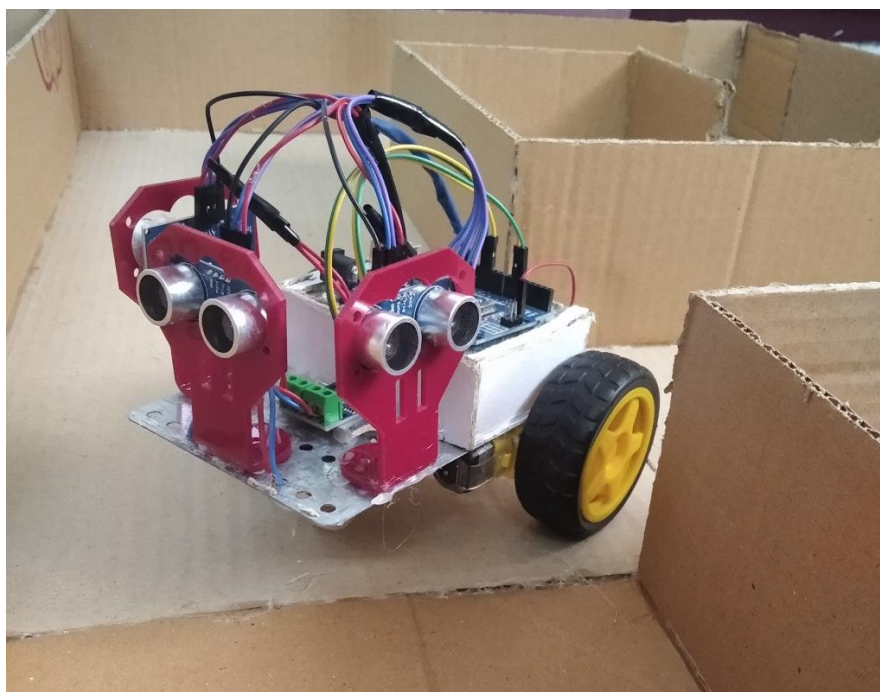
- موتور گیربکسی ۱۰۰RPM
- باتری لیتیوم یونی ۱۸۶۵۰
- باتری کتابی ۹V
- ماژول درایور L9110S
- پلتفرم آردوینو
- ماژول اولتراسونیک
- ماژول وایرلس nRF24L01

دو نمونه ربات برای پروژه ساخته شده نمونه ۴ چرخ و نمونه ۳ چرخ که در ادامه تصویر آن ها گذاشته

شده است.



شکل ۵-۲: پیاده سازی نهایی ربات در محیط واقعی نمونه ۴ چرخ



شکل ۵-۳: پیاده سازی نهایی ربات در محیط واقعی نمونه ۳ چرخ

۵-۳- الگوریتم نوشته شده در نرم افزار Arduino

چون در پروژه هدف ارسال مسیر صحیح به ربات همکار است، بنابراین از الگوریتم هایی که نمی‌تواند مسیر را رهگیری کند نمی‌توان استفاده کرد الگوریتم هایی نظیر دنبال کردن دیوارها الگوریتم های تصادفی و...

همانطور که در قسمت های قبل گفته شد، ربات باید بتواند مسیر را رهگیری کند و در پشته ^{۳۹} ذخیره کند تا در انتهای مسیر ماز یعنی وقتی که به هدف رسید بتواند مسیر ذخیره شده در پشته را به ربات همکار ارسال کند.

برای همین برای این پروژه از ترکیب الگوریتم پیمایش گراف ها به روش BFS و الگوریتم FLOOD برای استفاده شده است.



```
File Edit Sketch Tools Help
***
 * Cooperation of Multiple Robots to Solve Maze Tasks
 * The circuit:
 *   1.Arduino Uno
 *   3.0c Motor 100RPM
 *   2.L9110H
 *   3.Ultrasonic
 *   4.HCF40101
 * Created At 20/12/2021
 * By Amirhossein Bahmani & Mohsen Bahimi
 */

//Initialize WIFI Module*/
#include <SPI.h>
#include <RF24.h>
#include <RF24.h>
#include <printf.h>

RF24 radio(9, 0); // (CE, CS) for ARD Nano
#define R01 5 // Motor A pins
#define R02 6
#define R01 10 // Motor B pins
#define R02 11

#define PWM_RIGHT 200
#define PWM_LEFT 200
#define D01 11

/**
 * Initialize all ultrasonics
 */
int trigPinLeft=A5;
int echoPinLeft=A4;
int trigPinRight=A1;
int echoPinRight=A0;
int trigPinFront=A3;
int echoPinFront=A2;
//
long duration;
int distance;
int cell=-1;
bool isFindCorrectPath=false;
bool deadend=false;
int send_count=0;
//-----end of code-----
```

شکل ۵-۴: محیط نرم افزار Arduino

توضیحات الگوریتم نوشته شده:

برای اینکه بتوانیم مسیر درست را در stack ذخیره کنیم و درواقع نقشه مسیر صحیح را بدست آوریم از یک آرایه به عنوان path استفاده شده است چون ماز انتخابی ۳۶ خانه دارد ما از یک آرایه ۳۶ خانه‌ای برای اینکار استفاده کردیم که جهت های حرکت ربات در خانه هایی که ربات در آن قرار می‌گیرد در این آرایه ذخیره می‌شود. وقتی ربات بین دوراهی قرار می‌گیرد برای اینکه مسیر های پیش رو ربات پیش بینی شود آرایه‌ی دیگری با عنوان Mode قرار داده شده که مقدار اولیه آن نال هست، و وقتی ربات در دوراهی قرار گرفت شماره خانه مورد نظر و نوع دو راهی با کارکترهای A (چپ و راست)، B (راست و مستقیم)، C (چپ و مستقیم) در این آرایه ذخیره می‌شود و همچنین در آرایه دیگری با عنوان two_way موقعیت هایی که ربات در دوراهی قرار گرفته در آن ذخیره می‌شود، سپس به وسیله تابع رندوم، ربات بصورت تصادفی یک مسیر را برای حرکت انتخاب می‌کند، که جهت انتخابی ربات به همراه موقعیت ربات در آرایه دیگری با عنوان cur_dir ذخیره می‌شود. در این حالت ۲ سناریو پیش می‌آید:

سناریو اول: مسیر انتخابی ربات صحیح باشد و مانعی بر سر راه ربات نباشد که ساده ترین حالت ممکن است و ربات به صورت عادی ادامه مسیر را طی می‌کند.

سناریو دوم: که چالش برانگیز است، این است که مسیر انتخابی ربات بن بست باشد که ربات باید مسیر طی شده را باز گردد و مسیر دیگری را انتخاب کند خب برای اینکار چون موقعیت حالت قبل ربات را ذخیره کرده بودیم ربات را به موقعیت قبلی برمیگردانیم و همچنین چون مسیر های طی شده اشتباه هستند باید از stack خارج شوند و همچنین جهت انتخابی جدید با جهت قبلی جایگزین شود، که ما این کار را در تابع turn_back() انجام دادیم به این صورت فقط مسیر های صحیح در آرایه Path ذخیره می‌شود و همینطور موقعیت های مشابه نیز به همین صورت طی می‌شود تا موقعی که ربات به انتهای مسیر ماز یعنی مقصد برسد سپس آرایه ذخیره شده که درواقع مسیر صحیح ماز از ابتدا تا مقصد است و

نقشه صحیح ماز ما است را برای ربات همکار از طریق وایفای ارسال می شود چون ماژول nRF24l01 نمی تواند بصورت همزمان بیشتر از ۳۲ بایت داده را ارسال کند به همین علت ما بصورت بیت به بیت داده ها را ارسال کردیم و در این حین ربات همکار منتظر دریافت اطلاعات می ماند بعد از دریافت اطلاعات و بررسی معتبر بودن اطلاعات، شروع به خواندن نقشه ارسالی می کند سپس براساس نقشه حرکت می کند تا به مقصد تعیین شده برسد.

کد نوشته شده در نرم افزار Arduino

کد نوشته شده در پیوست ۱ آورده شده است.

۴-۵- نتایج بدست آمده

الگوریتم در نرم افزار Arduino نوشته شده و از طریق شبیه ساز و ترمینال نرم افزار تست شده و نتایج بدست آمده با نتایج مورد انتظار پروژه همخوانی داشته است.

۵-۵- کارهایی که در ادامه می توان انجام داد

از جمله کارهایی که می توان انجام داد این است که الگوریتم نوشته شده را بر روی یکی از میکروهای Xmega پیاده سازی کرد و همچنین بجای موتورهای گیربکسی از موتور های dc با کیفیت بالاتر و یا از استپر موتور ها استفاده شود و در محیط واقعی تست انجام شود دوباره و نتایج بدست آمده با نتایج قبلی مقایسه شود. و همچنین می توان امکان پیدا کردن کوتاه ترین مسیر را نیز به الگوریتم نوشته شده اضافه کرد و کارهایی از این دست.

پیوست ۱: الگوریتم نوشته شده در نرم افزار Arduino

کد مربوط به ربات اصلی^{۴۰}:

```
/**
 * Main Robot
 * Cooperation of Multiple Robots to Solve Maze Tasks
 * The circuit:
 * 1.Arduino Uno
 * 2.Dc Motor 100RPM
 * 3.L9110S
 * 4.Ultrasonic HC-SR04
 * 5.Nrf24l01
 * Created At 20/12/2021
 * By Amirhossein Rahmani & Mohsen Rahimi
 */
```

```
/*Initilize WIFI Madule*/
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <printf.h>
RF24 radio(9, 8); // (CE, CSN) for ARD Nano
```

```
/* Initilize Driver L9110s */
#define MA1 5 // Motor A pins
#define MA2 6
#define MB1 10 // Motor B pins
#define MB2 11
```

⁴⁰ Main Robot

```

#define PWM_RIGHT 200
#define PWM_LEFT 200
#define DST 11

/*initialize all ultrasonics*/
int trigPinLeft=A5;
int echoPinLeft=A4;
int trigPinRight=A1;
int echoPinRight=A0;
int trigPinFront=A3;
int echoPinFront=A2;
//
long duration;
int distance;
int cell=-1;
bool isFindCorrectPath=false;
bool deadend=false;
int send_count=0;
int data;
uint32_t path[36],two_way[36]={0,0,0,0,0,0,
                                0,0,0,0,0,0,
                                0,0,0,0,0,0,
                                0,0,0,0,0,0,
                                0,0,0,0,0,0,
                                0,0,0,0,0,0,
                                0,0,0,0,0,0};

/**
 * All modes that the robot must choose between that
 * mode : null,A:RL(Right and Left open),B:FR(Front and Right
open),C:LF(Front and Left open)
 * curr_dir: `F`:Forward,`R`:turn Right,`L`:turn Left
 */
char mode[36],curr_dir[36]={'\0','\0','\0','\0','\0','\0',

```

```

'\0','\0','\0','\0','\0','\0',
'\0','\0','\0','\0','\0','\0',
'\0','\0','\0','\0','\0','\0',
'\0','\0','\0','\0','\0','\0',
'\0','\0','\0','\0','\0','\0'};

```

```

const byte address[6] = {'R','x','A','A','A'};
uint32_t correct_path[36];
void setup() {
    Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
    pinMode(trigPinFront,OUTPUT);
    pinMode(echoPinFront,INPUT);
    pinMode(trigPinRight,OUTPUT);
    pinMode(echoPinRight,INPUT);
    pinMode(trigPinLeft,OUTPUT);
    pinMode(echoPinLeft,INPUT);

    pinMode(MA1, OUTPUT);
    pinMode(MA2, OUTPUT);
    pinMode(MB1, OUTPUT);
    pinMode(MB2, OUTPUT);

    analogWrite(MA1, LOW);
    analogWrite(MA2, LOW);
    analogWrite(MB1, LOW);
    analogWrite(MB2, LOW);

    //setup wifi
    radio.begin();
    radio.setAutoAck(1);
    radio.enableAckPayload();
    radio.setRetries(5,15);
}

```

```

    radio.openWritingPipe(address);
    radio.setPALevel(RF24_PA_MAX);
    radio.stopListening();
}

void loop() {
    Stop();
    delay(5000);
    //just front open
    if(isLeftWall() && isRightWall() && !isFrontWall()){
        forward();
        path[++cell]=1;
        Serial.println("forward first");
        delay(100);
        Stop();
    }

    //just right open
    else if(isLeftWall() && !isRightWall() && isFrontWall()){
        path[++cell]=2;
        turnRight();
        delay(50);
        forward();
        delay(100);
        Serial.println("right");
    }

    //just left open
    else if(!isLeftWall() && isRightWall() && isFrontWall()){
        path[++cell]=3;
        turnLeft();
        delay(50);
        forward();
    }
}

```

```

    delay(100);
    Serial.println("left");
}

//left and right open
else if(!isLeftWall() && !isRightWall() && isFrontWall()){
    int rand_=random(1,3);
    //choose random way between left and right
    if(rand_==1){
        path[++cell]=2;
        curr_dir[cell]='R';
        turnRight();
        delay(50);
        forward();
        delay(100);
        Serial.println("right 1");
    }
    if(rand_==2){
        path[++cell]=3;
        curr_dir[cell]='L';
        turnLeft();
        delay(50);
        forward();
        delay(100);
        Serial.println("left..");
    }
    two_way[cell]=1;
    mode[cell]='A';//LR
    Stop();
    delay(70);
}

```

```

//left and front open
else if(!isLeftWall() && isRightWall() && !isFrontWall()){
    int rand_=random(1,3);
    //choose random way between left and front
    if(rand_==1){
        path[++cell]=1;
        curr_dir[cell]='F';
        forward();
        delay(100);
        Serial.println("forward..");
    }
    if(rand_==2){
        path[++cell]=3;
        curr_dir[cell]='L';
        turnLeft();
        delay(50);
        forward();
        delay(100);
        Serial.println("left..");
    }
    two_way[cell]=1;
    mode[cell]='C';//LF
    Stop();
    delay(70);
}

//right and front open
else if(isLeftWall() && !isRightWall() && !isFrontWall()){
    int rand_=random(1,3);
    //choose random way between right and front
    if(rand_==1){
        path[++cell]=1;

```

```

        curr_dir[cell]='F';
        forward();
        delay(100);
        Serial.println("forward..");
    }
    if(rand_==2){
        path[++cell]=2;
        curr_dir[cell]='R';
        turnRight();
        delay(50);
        forward();
        delay(100);
        Serial.println("right 2");
    }
    two_way[cell]=1;
    mode[cell]='B';//FR
    Stop();
    delay(70);
}

//Deadend left and right and front wall
else if(isDeadend()){
    Serial.println("Deadend");
    turnRight();
    //turn back 180deg
    delay(2000);
    Serial.println("turn back 180deg");
    Stop();
    deadend=true;
}

//turn back if is deadend

```



```

if(deadend){
    turn_back();
    delay(1000);
}

//right and left and front open all dir open
if(isAllDirOpen()){
    Serial.println("all dir open");
    Serial.println("forward");
    forward();
    delay(4000);
    Stop();
    path[++cell]=1;
    if(isAllDirOpen()){
        Stop();
        delay(2000);
        Serial.println("End maze..");
        isFindCorrectPath=true;
        //End maze and send path to another robot
        copy(path,correct_path,36);
    }
}

/**
 * after solve maze and find correct path send correct path to Fellow
robot
 */
while(isFindCorrectPath){
    Stop();
    Serial.println("Stop Motor");
    if(send_count==0){
        send_correct_path_to_fellow_robot();
    }
}

```

```

    }else{
        Serial.println("Already Sended.");
    }
    delay(400);
}
}

void turn_back(){
    for(;two_way[cell]!=1;cell--){
        if(path[cell]==1){
            forward();
            Serial.println("turn back :forward");
            delay(100);
            Stop();
        }else if(path[cell]==2){
            turnLeft();
            delay(50);
            forward();
            delay(100);
            Serial.println("turn back: left");
        }else if(path[cell]==3){
            turnRight();
            delay(50);
            forward();
            delay(100);
            Serial.println("turn back: right");
        }
        path[cell]=0;
        delay(5000);
        Stop();
    }
    //two way

```

```

if(two_way[cell]==1){
    if(mode[cell]=='A' && curr_dir[cell]=='L'){
        forward();
        Serial.println("two turn back: forward");
        delay(100);
        Stop();
        path[cell]=2;
    }
    if(mode[cell]=='A' && curr_dir[cell]=='R'){
        forward();
        Serial.println("two turn back: forward");
        delay(100);
        Stop();
        path[cell]=3;
    }
    //left and front open
    if(mode[cell]=='C' && curr_dir[cell]=='F'){
        turnRight();
        delay(50);
        forward();
        delay(100);
        path[cell]=3;
        Serial.println("two turn back : right");
    }
    if(mode[cell]=='C' && curr_dir[cell]=='L'){
        turnLeft();
        delay(50);
        forward();
        delay(100);
        Serial.println("turn back: left");
        path[cell]=1;
    }
}

```

```

//right and front open
if(mode[cell]=='B' && curr_dir[cell]=='F'){
    turnLeft();
    delay(50);
    forward();
    delay(100);
    Serial.println("two turn back: left");
    path[cell]=2;
}
if(mode[cell]=='B' && curr_dir[cell]=='R'){
    turnRight();
    delay(50);
    forward();
    delay(100);
    Serial.println("two turn back: right");
    path[cell]=1;
}
}
Stop();
delay(180);
deadend=false;
exit;
}

void forward() {           //function of forward
    analogWrite(MA1, LOW);
    analogWrite(MA2, PWM_LEFT);
    analogWrite(MB1, LOW);
    analogWrite(MB2, PWM_RIGHT);
}

```

```

void backward() {           //function of backward
    analogWrite(MA1, HIGH);
    analogWrite(MA2, 0);
    analogWrite(MB1, HIGH);
    analogWrite(MB2, 0);
}

void Stop() {               //function of stop
    analogWrite(MA1, LOW);
    analogWrite(MA2, LOW);
    analogWrite(MB1, LOW);
    analogWrite(MB2, LOW);
}

void turnLeft() {          //function of turn left
    analogWrite(MA1, PWM_LEFT);
    analogWrite(MA2, LOW);
    analogWrite(MB1, LOW);
    analogWrite(MB2, PWM_RIGHT);
}

void turnRight() {         //function of turn right
    analogWrite(MA1, LOW);
    analogWrite(MA2, PWM_LEFT);
    analogWrite(MB1, PWM_RIGHT);
    analogWrite(MB2, LOW);
}

/**
 * get distance(cm) by ultarsonic sensor
 */
int getDuration(int trigPin,int echoPin){

```

```

    digitalWrite(trigPin, LOW);
    delayMicroseconds(5);
    digitalWrite(echoPin, LOW);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    duration=pulseIn(echoPin, HIGH);
    return duration;
}

int rightSensor(){
    long duration=getDuration(trigPinRight,echoPinRight);
    return (duration*0.034/2);
}

int leftSensor(){
    long duration=getDuration(trigPinLeft,echoPinLeft);
    return (duration*0.034/2);
}

int frontSensor(){
    long duration=getDuration(trigPinFront,echoPinFront);
    return (duration*0.034/2);
}

bool isLeftWall(){
    if(leftSensor()<DST){
        return true;
    }else{
        return false;
    }
}

bool isRightWall(){
    if(rightSensor()<DST){

```

```

        return true;
    }
    return false;
}

bool isFrontWall(){
    if(frontSensor() < DST){
        return true;
    }
    return false;
}

bool isAllDirOpen(){
    if(!isLeftWall() && !isRightWall() && !isFrontWall()){
        return true;
    }
    else
        return false;
}

bool isDeadend(){
    if(isFrontWall() && isRightWall() && isLeftWall()){
        return true;
    }else{
        return false;
    }
}

void send_correct_path_to_fellow_robot(){
    Serial.println("Sending...");
    for (int j = 0; j < 32; j++) {
        data=correct_path[j];
    }
}

```

```

        radio.write(&data, sizeof(data));
        Serial.println(data);
        delay(1000);
    }
    Serial.println("sended succeeded");
    send_count=1;
}

// Function to copy 'len' elements from 'src' to 'dst'
void copy(uint32_t* src, uint32_t* dst, int len) {
    memcpy(dst, src, sizeof(src[0])*len);
}

```

کد مربوط به ربات همکار^{۴۱}:

```

/**
 * Fellow Robot
 * Cooperation of Multiple Robots to Solve Maze Tasks
 * The circuit:
 * 1.Arduino Uno
 * 2.Dc Motor 100RPM
 * 3.L9110S
 * 4.Ultrasonic HC-SR04
 * 5.Nrf24l01
 * Created At 20/12/2021
 * By Amirhossein Rahmani & Mohsen Rahimi
 */

/*Initilize WIFI Madule*/
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

```

⁴¹ fellow-robot


```

#include <printf.h>

RF24 radio(9, 8); // (CE, CSN) for ARD Nano
const byte address[6] = {'R','x','A','A','A'};
#define MA1 5 // Motor A pins
#define MA2 6
#define MB1 10 // Motor B pins
#define MB2 11

#define PWM_RIGHT 200
#define PWM_LEFT 120

bool start_=false;
bool is_read_data=false;
int j=-1;
int path[32];
int data;
int count=0;

/**
 * path=1 --> move forward
 * path=2 --> turn right
 * path=3 -->turn left
 */
void setup() {
    Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
    pinMode(MA1, OUTPUT);
    pinMode(MA2, OUTPUT);
    pinMode(MB1, OUTPUT);
    pinMode(MB2, OUTPUT);

    radio.begin();

```

```

    radio.setAutoAck(1);
    radio.enableAckPayload();
    radio.setRetries(5,15);
    radio.openReadingPipe(1,address); // for Ard. NANO
    radio.setPALevel(RF24_PA_MAX);
    radio.startListening();
}

void loop() {
    if(!is_read_data){
        while(isDataAvailable()){
            read_data();
            if(count>32){
                is_read_data=true;
                if(isValidData()){
                    start_=true;
                    Serial.println("Data read and is valid.");
                }else{
                    Serial.println("Data read and is not valid check your
connection.");
                }
            }
            count++;
        }
    }

    if(start_){
        for(int i=0;i<36;i++){
            if(path[i]==1){
                forward();
                delay(100);
                Stop();
            }
        }
    }
}

```

```

    }

    if (path[i]==2) {
        turnRight();
        delay(50);
        forward();
        delay(100);
        Stop();
    }

    if (path[i]==3) {
        turnLeft();
        delay(50);
        forward();
        delay(100);
        Stop();
    }

    delay(4000);
    Stop();
}

start_=false;
//end maze
}

}

void forward() {           //function of forward
    analogWrite(MA1, LOW);
    analogWrite(MA2, PWM_LEFT);
    analogWrite(MB1, LOW);
    analogWrite(MB2, PWM_RIGHT);
}

```

```

void backward() {           //function of backward
    analogWrite(MA1, HIGH);
    analogWrite(MA2, 0);
    analogWrite(MB1, HIGH);
    analogWrite(MB2, 0);
}

void Stop() {               //function of stop
    analogWrite(MA1, LOW);
    analogWrite(MA2, LOW);
    analogWrite(MB1, LOW);
    analogWrite(MB2, LOW);
}

void turnLeft() {           //function of turn left
    analogWrite(MA1, PWM_LEFT);
    analogWrite(MA2, LOW);
    analogWrite(MB1, LOW);
    analogWrite(MB2, PWM_RIGHT);
}

void turnRight() {          //function of turn right
    analogWrite(MA1, LOW);
    analogWrite(MA2, PWM_LEFT);
    analogWrite(MB1, PWM_RIGHT);
    analogWrite(MB2, LOW);
}

bool isDataAvailable(){
    if (radio.available()) {
        Serial.print("Received Data : ");
    }
}

```

```

        return true;
    }
    Serial.println("Not Receiving !!!");
    return false;
}

void read_data(){
    if (j<32){
        radio.read(&data, sizeof(data));
        path[++j]=data;
        Serial.println(path[j]);
        delay(1000);
    }
}

bool isValidData(){
    int count=0;
    for(int i=0;i<32;i++){
        if(path[i] > 0 && (path[i]==1 || path[i]==2 || path[i]==3)){
            return true;
        }
        if(path[i]==0){
            count++;
        }
        delay(500);
    }
    if(count>30){
        return false;
    }
}

```

- [1] <https://en.wikipedia.org/wiki/Micromouse>
- [2] https://en.wikipedia.org/wiki/Graph_theory
- [3] <https://blog.faradars.org/graph-data-structure/>
- [4] <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>
- [5] <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>
- [6] https://en.wikipedia.org/wiki/Maze-solving_algorithm
- [7] <https://www.geeksforgeeks.org/flood-fill-algorithm/>
- [8] https://www.tutorialspoint.com/data_communication_computer_network/wireless_transmission.htm
- [9] <https://roboworld.com/how-to-select-the-right-motor-for-your-robot/>
- [10] <https://www.baeldung.com/java-solve-maze>
- [11] <https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/>
- [12] Nordic Semiconductor ASA- Single chip 2.4 GHz Transceiver nRF24L01- Vestre Rosten 81, N-7075 Tiller, Norway, Norway, March 2006

Abstract:

The quick development of technology leads us to careful planning for best selection. These technologies and innovations in the lives of individuals making their work easier. in this paper, autonomous maze solving robot is developed with independent mapping and localization skill. in first step, the maze solving bot is designed with three ultrasonic distance sensor which is used for wall detection to avoid collision and obstacle detection. Also, it expects to use robot where an environment unreachable for human. In addition, there are also places where use of robots is the only way to achieve a goal. We have successfully implemented the maze solution ability on the robot. The result of the experiment was that the robot could successfully solve the maze with the walls and also save the correct path in its memory and send the saved path to the fellow robot at the end of the path.in this design, the possibility of sending positions to other robots is embedded.

Keywords:

Maze – Micro mouse – Algorithm – Graph – Graph navigation – Micro Controller – ATmega 328 – Sensors



Shahid Rajaee
Teacher Training University
Faculty of Electrical Engineering
Department of Electronic
B.Sc. Thesis

Title:

Cooperation of Multiple Robots to Solve Maze Tasks

Supervisor:

Dr. Nasour Bagheri

By:

Mohsen Rahimi & Amirhossein Rahmani

February 2022