دانشکده شهید منتظری

گروه برق و کنترل

پایان نامه کارشناسی رشته برق گرایش کنترل

عنوان

طراحی و پیاده سازی کنترلر ربات اسکارا

استاد راهنما

جناب دکتر بهلوری

نگارش

محمد شریکیان
فرزاد صبور

خرداد 1405

فهرست مطالب

فصل اول: معرفی و بررسی ربات های صنعتی

فصل دوم: معرفی و بررسی ربات اسکارا

فصل سوم: معرفی و بررسی اجزای ربات اسکارا

فصل چهارم: معرفی و بررسی کنترلر ربات اسکارا

فصل اول: معرفی و بررسی ربات های صنعتی

شکل ۱- ۱- معرفی و بررسی ربات های صنعتی

۱-۱. ربات صنعتی چیست؟

ربات‌های صنعتی ماشین‌های خودکاری هستند که با داشتن قابلیت‌های کنترل اتوماتیک، برنامه‌پذیری و چند منظوره بودن، بیشترین جایگاه را در صنایع گوناگون به خود اختصاص داده است. این ربات‌ها معمولاً دارای چندین محور حرکتی هستند و می‌توانند وظایفی مانند جوشکاری، مونتاژ، بسته‌بندی و بازرسی را با دقت و سرعت بالا انجام دهند.

این ربات‌ها اغلب در صنعت و تولید انبوه کاربرد دارند و می‌توانند به طور خودکار وظایف تکراری و پیچیده را انجام دهند همچنین آنها وابستگی انسان را کاهش می‌دهند و بهره‌وری و ظرفیت فرآیندی را نیز افزایش می‌دهند.

شکل ۱- ۲- ربات صنعتی چیست؟

۱-۲. انواع ربات های صنعتی

ربات های صنعتی به انواع مختلفی تقسیم می‌شوند که هر کدام برای کاربردهای خاصی طراحی شده‌اند. در این بخش به بررسی مهم‌ترین انواع ربات های صنعتی می‌پردازیم:

شکل ۱- ۳- انواع ربات های صنعتی

۱-۲-۱. ربات بازویی (Arm Robot)

ربات های بازویی یا بازوهای رباتیک، متنوع ترین و پرکاربردترین نوع از انواع ربات های صنعتی هستند. آنها از یک سری اتصالات چرخشی تشکیل شده‌اند که توسط موتورهای سروو کار می‌کنند همچنین این مفاصل، انعطاف‌پذیری بازوی انسان را تقلید می‌کنند و فوق‌العاده سریع هستند.

شکل ۱- ۴- ربات بازویی

این ربات ها به طور معمول، می‌توانند سه تا شش محور حرکتی داشته باشند که به آنها اجازه می‌دهد در فضاهای محدود حرکت کنند و کارهای پیچیده را انجام دهند.

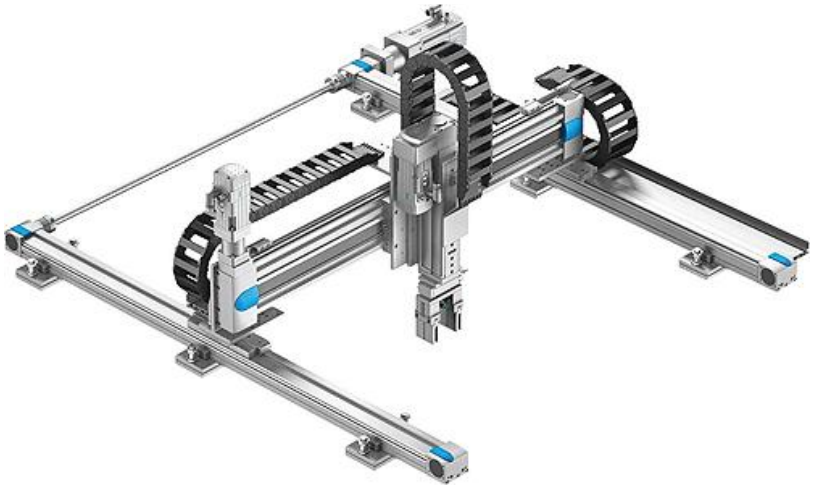
۱-۲-۲. ربات اسکارا (SCARA Robot)

ربات اسکارا یکی از انواع ربات های بازویی می باشد که بدلیل طراحی خاص آن در دسته بندی مجزایی از بازو های رباتیک شناخته می شوند. ربات اسکارا دارای انعطاف‌پذیری کمتری نسبت به ربات های بازویی است، اما در برخی کاربردها مانند مونتاژ و جابه‌جایی قطعات کوچک بسیار کارآمدتر هستند.

شکل ۱- ۵- ربات اسکارا

۱-۲-۳. ربات کارتزین (Cartesian Robot)

ربات کارتزین که به عنوان ربات‌های خطی یا گانتری[1] نیز شناخته می‌شود، دارای سه محور خطی است که به آن اجازه می‌دهد در یک فضای سه‌بعدی حرکت کند. هر محور، توسط یک محرک خطی مانند بال اسکرو یا موتور خطی هدایت می‌شود که امکان حرکت دقیق و مستقل را فراهم می‌کند.



شکل ۱- ۶- ربات کارتزین

این بازوها معمولاً در کاربردهایی که نیاز به دقت بالا و حرکات خطی دارند، مورد استفاده قرار می‌گیرند.

[1] Gantry

۱-۲-۴. ربات دلتا یا موازی[1] (Delta Robot)

ربات دلتا گونه‌ای از انواع ربات های صنعتی هستند که دارای چندین بازوی موازی متصل به یک جایگاه مثلثی شکل می باشند. این جایگاه در بالای محل کار نصب می‌شود و هر بازو میزبان یک سروو موتور با گشتاور بالا می‌باشد. شفت موتور به بازویی متصل است که در جهت عمود بر محور چرخش موتور امتداد دارد.

شکل ۱- ۷- ربات دلتا

این بازوها به دلیل سرعت و دقت بالا، درکاربردهایی مانند بسته‌بندی و مونتاژ قطعات کوچک بسیار مفید هستند.

شکل ۱- ۸- کاربرد ربات دلتا

[1] Parallel Robot

۱-۳. ویژگی های انواع ربات های صنعتی

ربات های صنعتی به دلیل قابلیت‌های فراوان و انعطاف‌پذیری بالا، مزایای بسیاری را برای صنایع مختلف به ارمغان می‌آورند. در این بخش به بررسی مزایای ربات های صنعتی می‌پردازیم:

- **افزایش بهره‌وری**

این ربات ها می‌توانند وظایف تکراری و زمان‌بر را با سرعت و دقت بالا انجام دهند که منجر به افزایش تولید و کاهش زمان تولید می‌شود.

- **بهبود کیفیت و دقت**

این ربات ها می‌توانند وظایفی مانند جوشکاری، مونتاژ و بسته‌بندی را با دقت بالا انجام دهند و خطاهای انسانی را به حداقل برسانند.

- **کاهش هزینه‌ها**

استفاده از ربات های صنعتی می‌تواند هزینه‌های تولید را کاهش دهد. این ربات ها نیاز به استراحت ندارند و می‌توانند به صورت مداوم کار کنند که منجر به کاهش هزینه‌های نیروی انسانی و افزایش بهره‌وری می‌شود.

- **افزایش ایمنی**

این ربات ها می‌توانند در محیط‌های خطرناک و با مواد خطرناک کار کنند و خطرات احتمالی را کاهش دهند.

- **انعطاف‌پذیری بالا**

این ربات ها می‌توانند به راحتی برنامه‌ریزی و تنظیم شوند تا وظایف مختلفی را انجام دهند و به تغییرات محیطی واکنش نشان دهند.

- **کاهش زمان توقف**

این ربات ها می‌توانند به صورت مداوم کار کنند و نیاز به تعمیرات و نگهداری کمتری دارند که منجر به افزایش بهره‌وری و کاهش هزینه‌ها می‌شود.

- **افزایش سرعت تولید**

ربات های صنعتی می توانند وظایف تولیدی را با سرعت بالا انجام دهند که منجر به افزایش سرعت تولید و کاهش زمان تولید می شود.

۱-۴. چالش های انواع ربات های صنعتی

ربات های صنعتی با چالش‌های متعددی مواجه هستند که می‌تواند تأثیرات قابل توجهی بر عملکرد و کارایی آنها داشته باشد. در این بخش به بررسی چالش‌های ربات های صنعتی می‌پردازیم:

- **هزینه‌های اولیه بالا**

خرید و نصب ربات های صنعتی نیاز به سرمایه‌گذاری قابل توجهی دارد که ممکن است برای بسیاری از شرکت‌ها و صنایع کوچک چالش‌برانگیز باشد.

- **نیاز به نظارت مداوم**

ربات های صنعتی نیاز به نظارت و نگهداری مداوم دارند تا از بروز خطاها و مشکلات مکانیکی جلوگیری شود. این نیاز به نظارت مداوم می‌تواند هزینه‌های اضافی برای شرکت‌ها ایجاد کند و بهره‌وری را کاهش دهد.

- **محدودیت در تطبیق‌پذیری**

ربات های صنعتی نمی‌توانند به اندازه دست انسان تطبیق‌پذیر باشند. این محدودیت در تطبیق‌پذیری می‌تواند در برخی کاربردها مشکل‌ساز باشد.

- **خطر بیکاری**

استفاده گسترده از ربات های صنعتی می‌تواند منجر به بیکاری کارگران انسانی شود. این موضوع می‌تواند تأثیرات اجتماعی و اقتصادی منفی داشته باشد و نیاز به برنامه‌ریزی دقیق برای مدیریت این تغییرات دارد.

- **توقف تولید در صورت خرابی**

در صورت خرابی ربات های صنعتی، کل خط تولید ممکن است متوقف شود. این توقف تولید می‌تواند هزینه‌های زیادی را به شرکت‌ها تحمیل کند و بهره‌وری را کاهش دهد.

- **محدودیت در ظرفیت بار**

این محدودیت‌ها می‌توانند در برخی کاربردها مشکل‌ساز باشند و نیاز به استفاده از ربات های قوی‌تر و گران‌تر را ایجاد کنند.

- **محدودیت‌های اتصال**

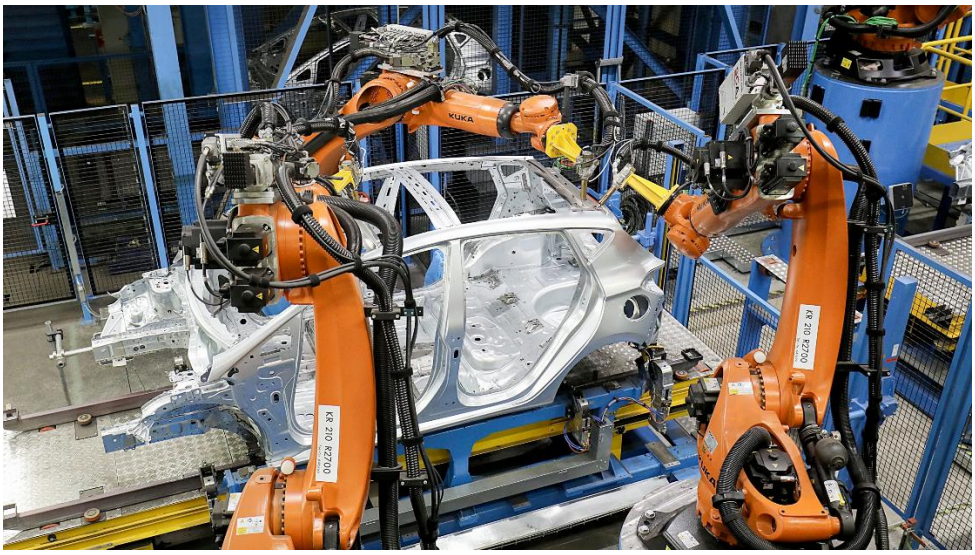
این محدودیت‌ها می‌توانند کارایی و انعطاف‌پذیری بازوها را کاهش دهند و نیاز به استفاده از تکنولوژی‌های پیشرفته‌تر را ایجاد کنند.

۱-۵. کاربردهای ربات های صنعتی

ربات های صنعتی به دلیل قابلیت‌های فراوان و انعطاف‌پذیری بالا، در صنایع مختلفی بکار گرفته می‌شوند. در این بخش به بررسی کاربردهای ربات های صنعتی می‌پردازیم:

- **صنعت خودروسازی**

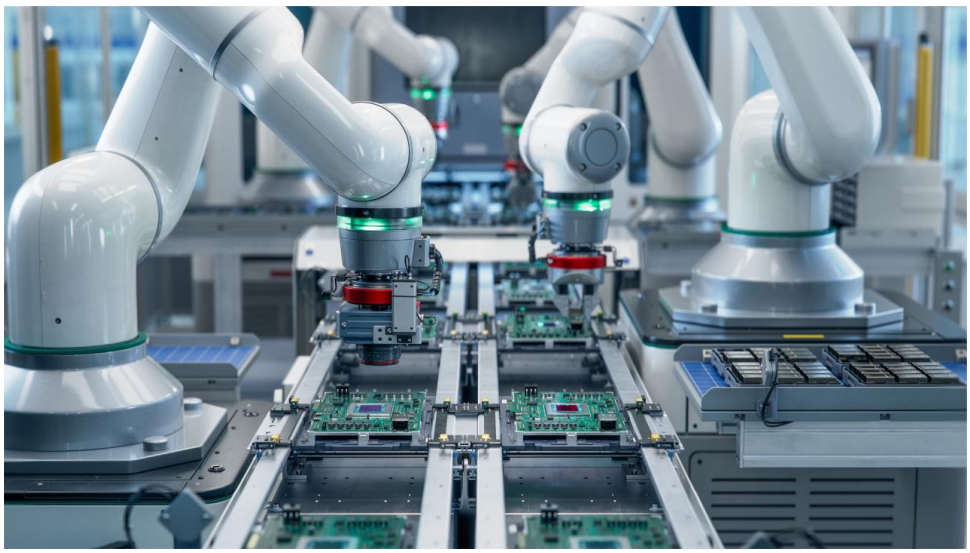
این ربات ها می‌توانند وظایفی مانند جوشکاری، مونتاژ، رنگ‌آمیزی و جابه‌جایی قطعات را با دقت و سرعت بالا انجام دهند.



شکل ۱- ۹- کاربرد ربات های صنعتی در صنعت خودروسازی

- **صنعت الکترونیک**

ربات های صنعتی برای مونتاژ قطعات کوچک و حساس بکار می‌روند. این ربات ها می‌توانند با دقت بالا و بدون خطا، قطعات الکترونیکی را مونتاژ کنند و کیفیت محصولات نهایی را بهبود بخشند.



شکل ۱- ۱۰- کاربرد ربات های صنعتی در صنعت الکترونیک

- **صنعت پزشکی**

این ربات ها می‌توانند در جراحی‌های پیچیده به جراحان کمک کنند و دقت و ایمنی را افزایش دهند. همچنین می‌توانند در تولید و بسته‌بندی داروها و تجهیزات پزشکی نیز بکار روند.

شکل ۱- ۱۱- کاربرد ربات های صنعتی در صنعتی پزشکی

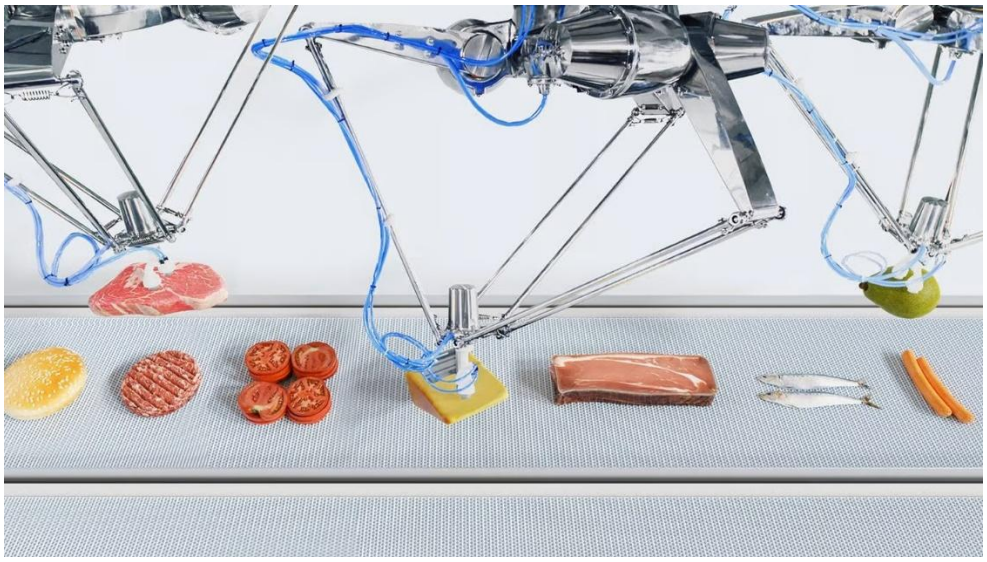
- **صنعت بسته‌بندی**

ربات های صنعتی برای جابه‌جایی و بسته‌بندی محصولات بکار می‌روند، این ربات ها می‌توانند با سرعت بالا و بدون خطا، محصولات را بسته‌بندی کنند و بهره‌وری را افزایش دهند.

شکل ۱- ۱۲- کاربرد ربات های صنعتی در صنعت بسته بندی

- **صنعت غذایی**

این ربات ها می‌توانند وظایفی مانند جابه‌جایی، بسته‌بندی و حتی پخت و پز را انجام دهند. همچنین استفاده از ربات های صنعتی در این صنعت باعث افزایش بهداشت و کاهش هزینه‌ها می‌شود.



شکل ۱- ۱۳- کاربرد ربات های صنعتی در صنایع غذایی

- **صنعت کشاورزی**

ربات های صنعتی برای وظایفی مانند برداشت محصول، کاشت و جابه‌جایی محصولات به کار می‌روند. این ربات ها می‌توانند با دقت و سرعت بالا، بهره‌وری کشاورزی را افزایش دهند.

شکل ۱- ۱۴- کاربرد ربات های صنعتی در صنعت کشاورزی

- **صنعت حمل و نقل**

ربات های صنعتی برای جابه‌جایی و بارگیری محموله‌ها به کار می‌روند. این ربات ها می‌توانند با دقت و سرعت بالا، بهره‌وری حمل و نقل را افزایش دهند.

شکل ۱- ۱۵- کاربرد ربات های صنعتی در صنعت حمل و نقل

فصل دوم: معرفی و بررسی ربات اسکارا

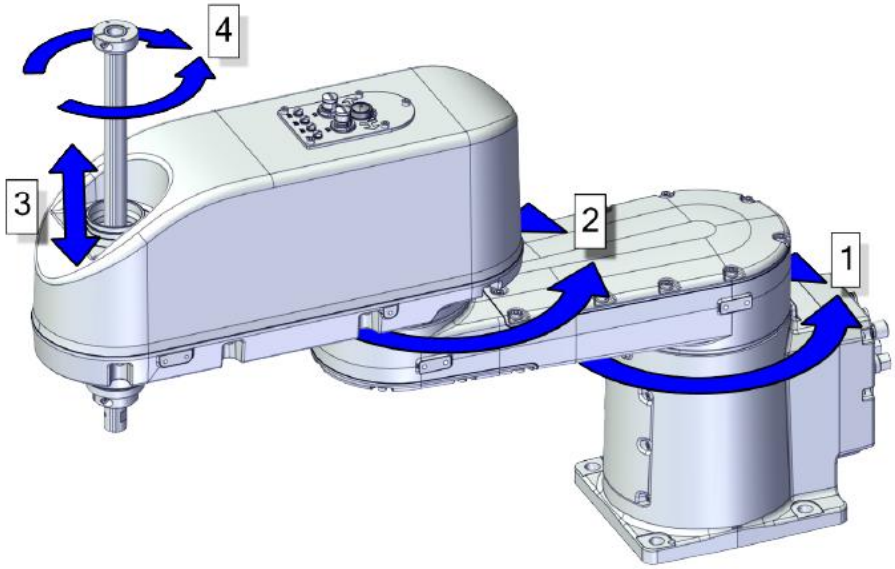
با ظهور فناوری رباتیک، جای تعجب نیست که صنایع بیشتری به ربات‌ها به عنوان راهی ساده برای افزایش بهره‌وری و ساده‌سازی اتوماسیون روی می‌آورند. از جمله این ربات‌ها می‌توان به ربات اسکارا اشاره کرد که یکی از محبوب‌ترین و کاربرپسندترین آنها است.

شکل ۲- ۱- معرفی و بررسی ربات اسکارا

۲-۱. ربات اسکارا چیست؟

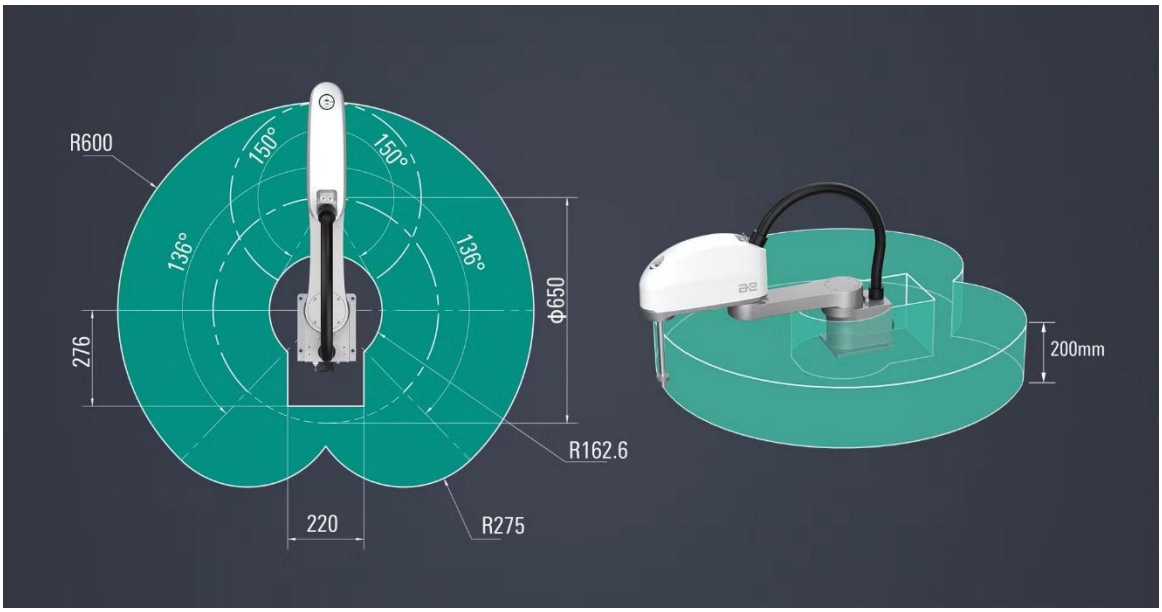
نام SCARA مخفف Selective Compliance Assembly Robot Arm است به معنی بازوی رباتیک مونتاژ با تطبیق پذیری انتخابی که به توانایی ربات برای حرکت آزادانه و حفظ صلابت در سه محور در حین سازگاری در محور نهایی اشاره دارد.

این ربات‌ها معمولاً دارای سه یا چهار محور هستند که به آن‌ها اجازه می‌دهد وظایف را با سرعت و دقت بالا انجام دهند.



شکل ۲- ۲- محورهای ربات اسکارا

ربات‌های اسکارا بدلیل دامنه حرکتی منحصر به فردشان، به ویژه در مختصات XY ، از دیگر ربات‌های صنعتی متمایز هستند. این ربات ها می توانند در امتداد محورهای X و Y حرکت کنند این بدین معنا است که آن‌ها می‌توانند بین چپ و راست (X) و بالا و پایین (Y) حرکت نمایند یا به زبان ساده، می‌توانند به صورت افقی در هر جهتی در فضای کاری خود حرکت کنند. همچنین با وجود برخی محدودیت‌ها می‌توانند به‌صورت عمودی حرکت نیز داشته باشند اما در این‌حالت محور عمودی ثابت خواهد ماند.



شکل ۲- ۳- دامنه حرکتی ربات اسکارا

این ربات‌های مونتاژی، قادر به انجام حرکات انعطاف‌پذیر و سخت هستند. این نوع انعطاف‌پذیری آنها را برای کارهایی مانند انتخاب و مکان‌یابی، مرتب‌سازی و مونتاژ مناسب می‌کند و استفاده از این نوع ربات، مثل داشتن یک کارگر خط مونتاژ فوق العاده کارآمد است که هیچ وقت خسته نمی‌شود و اشتباه نمی‌کند.

۲-۲. تاریخچه ربات اسکارا

ربات‌های اسکارا سابقه‌ای طولانی و قدیمی دارند. در سال ۱۹۷۷، پروفسور هیروشی ماکینو[1] از دانشگاه یاماناشی در سمپوزیوم بین‌المللی ربات‌های صنعتی در توکیو ژاپن شرکت کرد. در این رویداد، او شاهد یک اختراع انقلابی در ربات مونتاژ SIGMA بود.

ماکینو با الهام از اولین ربات مونتاژکننده، کنسرسیوم ربات اسکارا را تشکیل داد که شامل ۱۳ شرکت ژاپنی بود. هدف این گروه بهبود بیشتر ربات‌های مونتاژ از طریق تحقیقات اختصاصی بود.

کنسرسیوم به سرعت کارکرد و اولین نمونه اولیه آن‌ها از ربات اسکارا یک سال بعد و در سال ۱۹۷۸ انجام شد. آن‌ها ساخته خود را بر روی طیف وسیعی از کاربردهای صنعتی آزمایش کردند و طراحی موجود را بیشتر بهبود بخشیدند و نسخه دوم را دو سال بعد منتشر کردند.

شکل ۲- ۴- یکی از نمونه های اولیه ربات اسکارا

هنگامی که اولین ربات تجاری اسکارا در سال ۱۹۸۱ عرضه شد، از آن به‌عنوان یک طراحی رباتیک پیشگامانه استقبال شد. این ربات ارائه‌شده، نسبت به قیمت، عملکرد بسیار مطلوبی داشت و فرآیندهای تولید صنعتی را در سراسر جهان متحول کرد.

[1] Hiroshi Makino

شکل ۲- ۵- مزایا و معایب ربات اسکارا

۲-۳. ویژگی های ربات اسکارا در صنعت

ربات‌های اسکارا بدلیل قابلیت‌هایی که دارند، مزایای زیادی می‌توانند برای استفاده‌کنندگان داشته باشند. از جمله این مزایا می‌توان به موارد زیر اشاره کرد:

- **دقت بالا**

ربات اسکارا به دلیل دقت باورنکردنی شناخته می‌شود که آن را برای کارهایی که نیاز به مونتاژ دقیق دارند، عالی می‌کند. محور ثابت Z آن‌ها از حرکات غیرضروری بالا و پایین جلوگیری می‌کند و کمک قابل توجهی به این دقت می‌کند.

- **سرعت بالا**

ربات‌های اسکارا از سریع‌ترین ربات‌ها در دنیای ربات‌های صنعتی هستند و پس از ربات‌های دلتا در رتبه دوم از لحاظ سرعت قرار دارند که این سرعت بالا برای کارهای تولیدی که نیاز به پردازش سریع دارند مفید است.

- **فشرده بودن و حجم کم**

ربات اسکارا در مقایسه با سایر ربات‌ها نسبتاً کوچک است. این ربات محدوده عملیاتی کوچکی دارد که آنرا برای فضاهای کاری محدودتر منحصر به فرد می‌کند.

- **مقرون به صرفه**

یک ربات اسکارا، ارزش بسیار خوبی از نظر هزینه ارائه می‌دهد. اندازه جمع و جور آن به این معنی است که به مواد زیادی برای ساخت نیاز ندارد و قطعات متحرک کمتری نسبت به سایر انواع ربات (به عنوان مثال ربات‌های بازویی) دارد که همه این عوامل باعث کاهش هزینه‌ها می‌شود و آن‌ها را به انتخاب هوشمندانه‌ای برای تولیدکنندگان با بودجه کم، تبدیل می‌کند.

- **ادغام آسان**

ربات‌های اسکارا می‌توانند به راحتی در سیستم‌های خودکار موجود ادغام شوند و در کنار سایر ماشین‌ها به‌خوبی کار کنند. این ربات‌های چهار محوره، چه در حال برداشتن و قرار دادن، مونتاژ، مراقبت از ماشین‌ها یا توزیع مواد و غیره همه‌کاره‌ای هستند که کار را آسان‌تر و کارآمدتر می‌کنند.

- **برنامه نویسی و راه اندازی آسان**

علاوه بر این، برنامه‌نویسی این ربات‌ها نیز بسیار آسان است، به خصوص اگر از نرم افزار RoboDK به‌عنوان نرم‌افزار برنامه‌نویسی ربات خود استفاده می‌کنید. کتابخانه این نرم‌افزار شامل ده‌ها ربات محبوب اسکارا است.

۲-۴. چالش های ربات اسکارا در صنعت

با تمام این مزایا، همانطورکه در بالا ذکر شد، چالش های زیادی نیز می‌توانند برای استفاده‌کنندگان داشته باشند. از جمله این چالش ها را می‌توان به موارد زیر اشاره کرد:

- **دامنه حرکت محدود**

این نوع ربات به یک فضای کاری نسبتاً کوچک و خاص محدود می‌شود. همچنین به‌اندازه انواع دیگر ربات‌ها مانند ربات‌های صنعتی ۶ محوره انعطاف‌پذیر نیستند.

- **قابلیت های محدود**

از نظر قابلیت استفاده، آن‌ها بسیار عقب‌تر از ربات‌های مدرن (ربات‌های مشارکتی[1]) هستند که برای کار در کنار انسان‌ها طراحی شده‌اند و معمولاً ۶ محور ارائه می‌دهند.

- **ظرفیت بار محدود**

با وجود اینکه این ربات‌ها سرعت بالایی را ارائه می‌دهند؛ تمایل دارند در اندازه بار خود محدود باشند. بالاترین باری که ربات‌های اسکارا می‌توانند تحمل کنند در حدود ۳۰ تا ۵۰ کیلوگرم است، درحالی که بازوهای ربات‌های صنعتی ۶ محوره می‌توانند تا ۲۰۰۰ کیلوگرم را تحمل کنند.

این چالش ها انواع وظایفی را که ربات اسکارا برای آنها مناسب است محدود می‌کنند و آن‌ها را برای کارهایی که به انعطاف‌پذیری، دامنه حرکت یا سازگاری بیشتری نیاز دارند، نامناسب می‌سازند.

[1] Collaborative Robot - Cobot

۲-۵. کاربردهای ربات اسکارا در صنعت

ربات‌های اسکارا ماشین‌هایی هستند که به طور هدفمند ساخته شده‌اند و دارای مکانیک منحصر به فردی هستند که آن‌ها را برای کاربردهای خاص ایده‌آل می‌کند. این کاربردها عبارت‌اند از:

- **عملیات انتخاب و جاسازی**

وقتی صحبت از مقرون به صرفه بودن و سرعت در کارهای انتخاب و جابه‌جایی به میان می‌آید، ربات‌های اسکارا بی‌رقیب هستند. راه‌اندازی این نوع ربات فوق‌العاده آسان است و شما می‌توانید از آن‌ها انتظار داشته باشید که با استفاده از ابزارهای مختلف اثرگذار، محصولات را به طرز ماهرانه‌ای انتخاب کنند.

شکل ۲- ۶- فرآیند انتخاب و جابجایی توسط ربات اسکارا

- **مونتاژ**

ربات اسکارا نقش مهمی در تولید و خطوط مونتاژ، از جابه‌جایی مواد و مرتب‌سازی اقلام گرفته تا جایگذاری قطعات در کنار هم دارند. آن‌ها در صنایعی مانند الکترونیک که شامل مونتاژ طرح‌های پیچیده هستند، بسیار محبوب می‌باشند.

شکل ۲- ۷- فرآیند مونتاژ توسط ربات اسکارا

پرینتر سه‌بعدی

با معرفی رباتیک، پرینترهای سه‌بعدی روز به روز در صنایع مختلف به ویژه در تولید، محبوبیت بیشتری پیدا کرده است. در حالی که ربات‌های ۶ محوره اغلب می توانند برای پروژه‌های پرینت سه‌بعدی بزرگ‌تر استفاده می‌شوند، ربات‌های اسکارا برای کاربردهای پرینت سه‌بعدی در مقیاس کوچک مفید هستند.

شکل ۲- ۸- فرآیند پرینت سه بعدی توسط ربات اسکارا

بسته‌بندی و پالت سازی

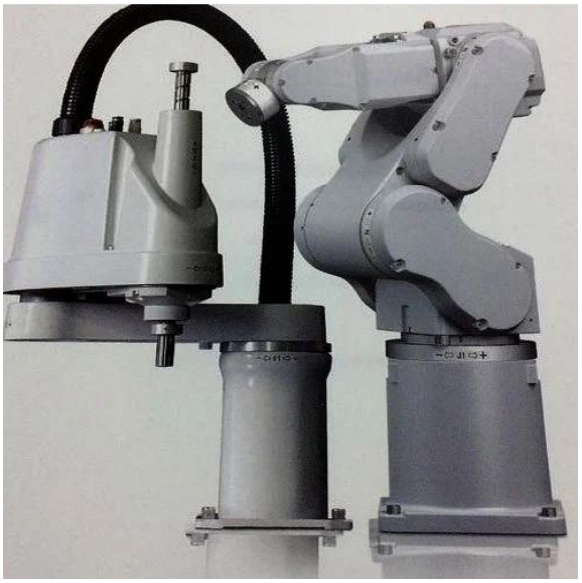
عملیات جابه‌جایی مواد، مانند بسته‌بندی، پالت‌گذاری و برچسب زدن، معمولاً یکنواخت هستند. ربات‌های اسکارا می‌توانند این وظایف تکراری را به طور خودکار انجام دهند و درعین حال از مدیریت راحت و دقیق فرآیند اطمینان حاصل کنند.

توزیع و پخش مواد

استفاده از چسب‌ها، رنگ‌آمیزی، آب‌بندی، کارهایی است که همه از ربات اسکارا بهره می‌برند. این ربات‌ها نه تنها این وظایف را بادقت و نتایج کلی بهتر انجام می‌دهند؛ بلکه به کاهش ضایعات و بهبود دقت کار در طول فرآیند توزیع کمک می‌کنند.

۲-۶. تفاوت ربات اسکارا با سایر انواع ربات‌ها

ربات‌های اسکارا از چند جهت با ربات‌های صنعتی استاندارد ۶ محوره متفاوت هستند. یکی از تفاوت‌های اصلی در نحوه طراحی آن‌ها است.



شکل ۲- ۹- تفاوت ربات اسکارا با سایر انواع ربات ها

واضح‌ترین تفاوت این است که یک ربات ۶ محوره دارای شش محور یا درجه آزادی است که به آن‌ها امکان می‌دهد در طیف گسترده‌ای از جهات حرکت کنند و وظایف خود را در محورهای مختصاتی مختلف انجام دهند؛ اما ربات‌های اسکار تنها چهار محور یا درجه آزادی دارند.

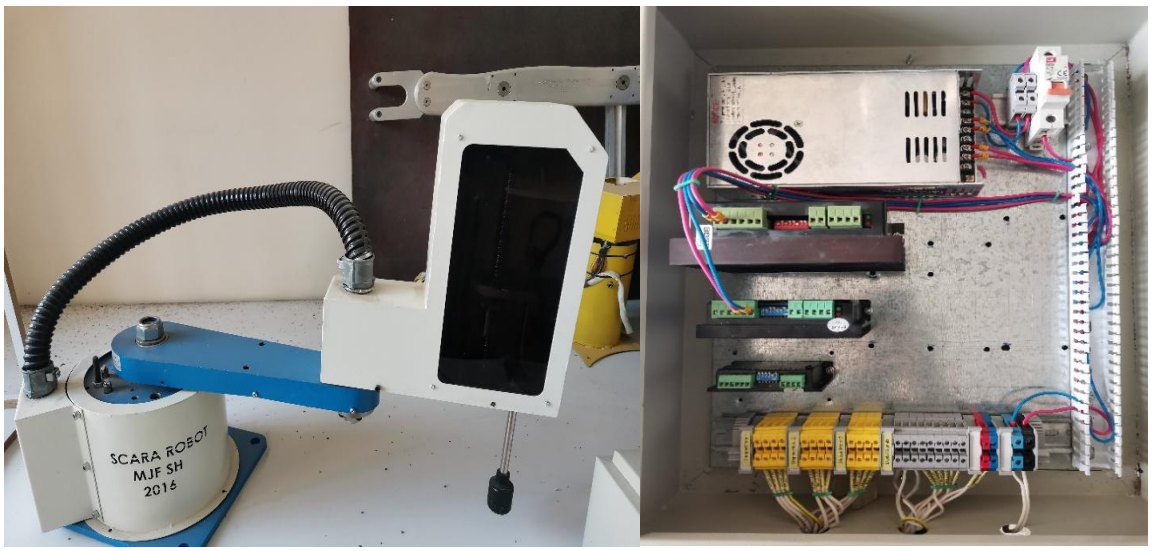
از آنجایی که این رباتها فقط چهار محور برای کنترل دارند، در مقایسه با سایر انواع ربات‌های صنعتی بسیار سریع هستند همچنین برنامه‌ریزی این ربات ها ساده می‌باشد؛ زیرا سینماتیک معکوس موجود در این ربات‌ها، بسیار ساده‌تر از بازوهای رباتیک صنعتی با ۶ درجه آزادی است.

یکی دیگر از تفاوت‌های کلیدی بین ربات‌های اسکارا و ربات‌های ۶ محوره، دامنه حرکت آن‌ها است. ربات‌های اسکارا دامنه حرکت محدودی در صفحه عمودی دارند؛ زیرا بازوی آن‌ها برای حرکت در درجه اول در صفحه افقی طراحی شده است که این باعث می‌شود برای کارهایی که نیاز به حرکات دقیق یا جابه‌جایی قطعات کوچک در یک فضای کاری خاص دارند، مناسب باشند؛ اما برای کارهایی که به دامنه وسیعی از حرکات عمودی یا حرکات پیچیده نیاز دارند، کمتر مناسب هستند.

علاوه بر این، ربات‌های اسکارا معمولاً کوچک‌تر و فشرده‌تر از ربات‌های ۶ محوره هستند و ادغام آن‌ها در فضاهای کاری کوچک‌تر را آسان‌تر می‌کند. همچنین سریع‌تر و دقیق‌تر از ربات‌های ۶ محور هستند که آن‌ها را برای کارهایی که نیاز به مونتاژ با سرعت بالا یا جابه‌جایی قطعات کوچک دارند، مناسب می‌کند.

فصل سوم: معرفی و بررسی اجزای ربات اسکارا

در فصل های پیشین بطور کامل درمورد ربات های صنعتی بخصوص ربات اسکارا بحث شد اما در این فصل قرار هست بطور تخصصی درمورد ربات ساخته شده در دانشگاه بپردازیم. ربات اسکارا بطور کلی از دو بخش مکانیک ربات و تابلو کنترل یا فرمان ربات تشکیل می شود.



شکل ۳- ۱- تابلو فرمان و مکانیک ربات اسکارا

۳-۱. مکانیک ربات اسکارا

مکانیک یا بدنه اصلی ربات اسکارا متشکل از سه محور می باشد که دو محور آن بصورت چرخشی یا مفصلی هستند که به یکدیگر متصل شده اند و محور نهایی نیز یک محور خطی می باشد که به عنوان محور Z ربات شناخته می شود.

در هر سه محور ربات از موتور های پله ای یا استپر موتور به عنواع محرک استفاده شده است که در محور اول و دوم به صورت مستقیم کوپل شده است و در محور نهایی نیز از یک بال اسکرو استفاده شده که به واسطه پولی و تسمه به موتور متصل شده است.

سنسورهای استفاده شده در این ربات به عنوان تنظیم کننده محدوده مجاز حرکت بازوهای ربات اسکارا می باشند که سنسور محدود کننده محور اول و آخر از نوع میکروسوئیچ می باشد و محور دوم نیز از سنسورهای القایی بهره گرفته شده است.

در ادامه به بررسی کامل تری از مشخصات اجزای مکانیک ربات اسکارا می پردازیم. در ابتدا به تعاریف و مفاهیم اولیه این اجزا و سپس مشخصات دقیق هر یک خواهیم پرداخت.

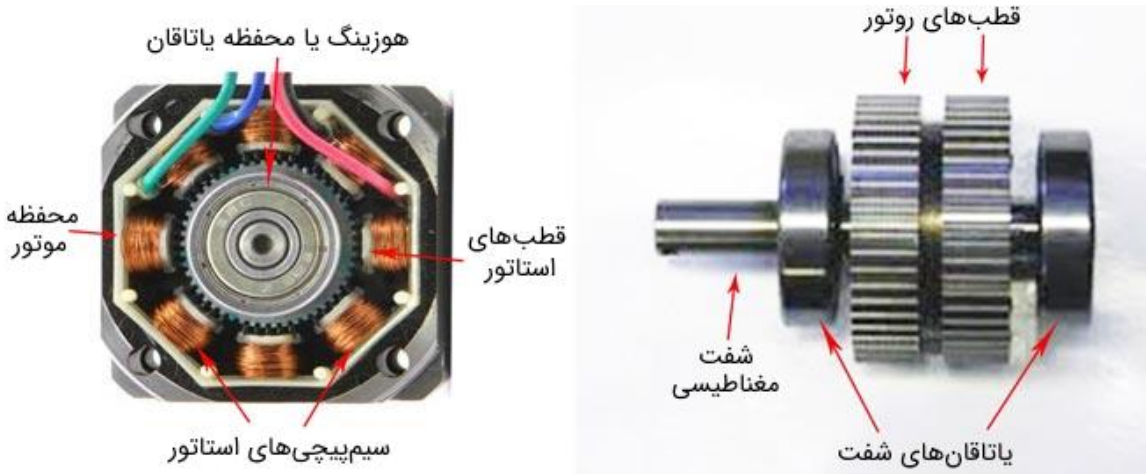
۳-۱-۱. معرفی و بررسی استپر موتور[1]

استپرموتورکه به نام موتور پله ای هم شناخته می شود یک موتور براشلس[2] یا بدون جاروبک است که یک دورکامل را به تعدادی پله تقسیم می کند و بدلیل وجود این پله ها توانایی حرکت گسسته یا پله به پله را دارد.

شکل ۳- ۲- معرفی و بررسی استپر موتور

استپر موتور از دو قسمت اصلی روتور و استاتور تشکیل شده است. روتور این موتورها معمولا یک آهن ربای مغناطیس دائم است. این روتورها توسط استاتورهای سیم پیچی شده احاطه شده‌اند.

هنگامی که سیم‌پیچ‌ها به صورت گام به گام و با یک ترتیب خاص فعال می‌شوند و جریان درون آن‌ها روانه می‌شود، باعث خواهد شد استاتورها مغناطیسه شده و قطب‌های الکترومغناطیسی بر روی آن ایجاد شود و روتور دروان کند. این قطب‌های مغناطیس شده اساس کار موتورهای استپر می‌باشند.

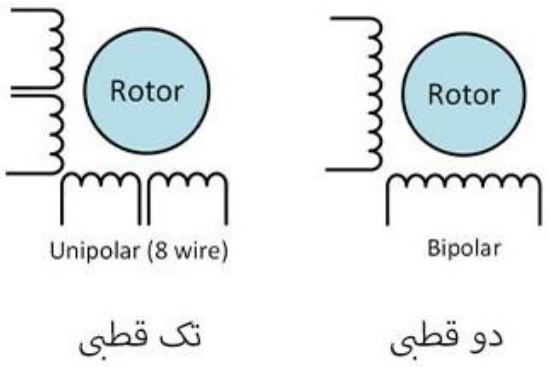


شکل ۳- ۳- ساختار داخلی استپر موتور

[1] Stepper Motor
[2] BLDC Motor

۳-۱-۱-۱. سیم بندی استپر موتور

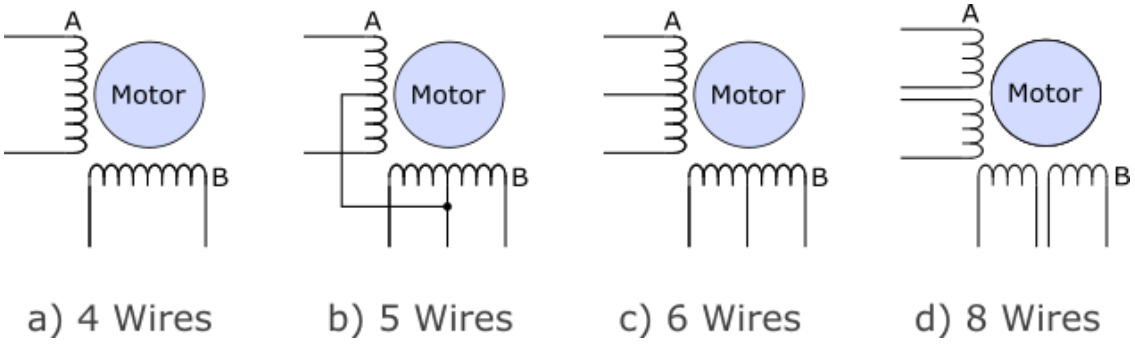
سیم بندی استپر موتور به دو دسته دو قطبی[1] و تک قطبی[2] تقسیم می شود. استپر موتورهای دو قطبی دو سیم پیچ مجزا دارند که انتهای هر یک از آن ها به صورت دو رشته سیم از موتور بیرون آمده است؛ حال آنکه در موتورهای تک قطبی چهار سیم پیچ مجزا وجود دارند.



شکل ۳- ۴- انواع سیم بندی استپر موتور

در استپر موتورهای تک قطبی شارش جریان در سیم پیچ‌ها همیشه در یک جهت است ولی در استپر موتورهای دو قطبی باید به طور مداوم جهت جریان در سیم پیچ‌ها عوض شود و همین امر راه‌اندازی این گونه موتورها را کمی دشوارتر می‌کند.

استپر موتورهای ۴ سیمه را تنها می‌توان به صورت دو قطبی یا Bipolar راه اندازی کرد ولی موتورهای ۵ و ۶ و ۸ سیمه به هر دو روش قابل راه‌اندازی هستند. قبل از راه‌اندازی هر نوع استپر موتور برای اجتناب از اتلاف وقت با استفاده از یک اهم متر نحوه اتصالات استپر موتور را تشخیص دهید.

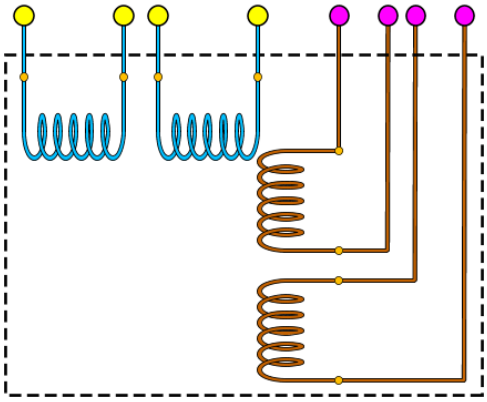


شکل ۳- ۵- انواع روش‌های اتصال سیم‌پیچ‌ها در استپر موتورها

[1] Bipolar
[2] Unipolar

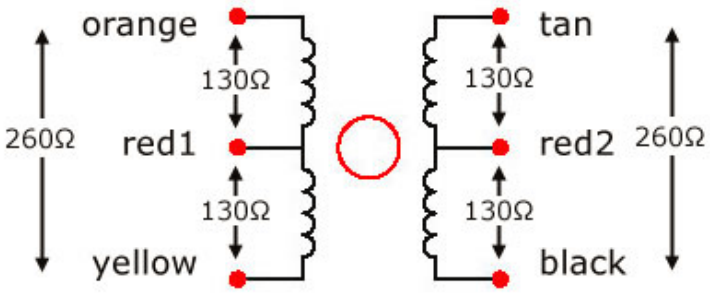
۳-۱-۱-۲. نحوه تشخیص اتصالات استپر موتور

در استپر موتورهای ۸ سیمه (شکل۳-۶) که انعطاف پذیر ترین شکل سیم پیچی را دارند دو سر هر چهار سیم پیچ در دسترس می‌باشد. برای اینکه بفهمید کدام دو سیم مربوط به کدام سیم پیچ است با استفاده از اهم متر دو سر هر زوج سیمی را به طور مجزا به اهم متر وصل کنید در صورت مشاهده مقاومتی بین ۱۰ تا ۲۰۰ اهم می‌توان گفت که آن دو سیم مربوط به یک سیم پیچ هستند در غیر این صورت اگر مقدار مقاومت بینهایت یا بسیار زیاد بود آن دو رشته سیم مربوط به یک سیم پیچ نیست.



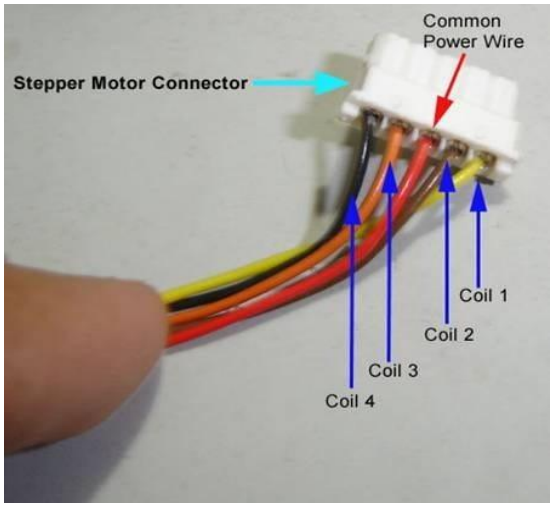
شکل ۳- ۶- استپر موتور ۸ سیمه

در استپر موتورهای ۶ سیمه سیم پیچ ها دو به دو با هم سری شده‌اند و سر وسط آن ها در دسترس می‌باشد. برای تشخیص نحوه اتصالات در استپر موتورهای ۶ سیمه نیز به طریق مشابه می‌توان با استفاده از اهم متر عمل کرد. فقط توجه شود که در این حالت مقدار مقاوت بین سر مشترک و دو سر یکی از سیم پیچ‌های سری شده نصف مقدار مقاومت بین دو سر دو سیم‌پیچ سری شده با هم می‌باشد. (شکل۳-۷)



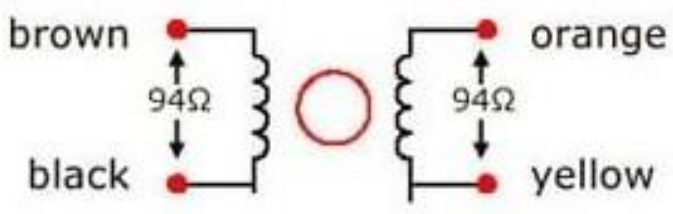
شکل ۳- ۷- نحوه تشخیص اتصالات در استپر موتور ۶ سیمه

در استپر موتورهای ۵ سیمه سرهای مشترک به هم وصل شده و به عنوان سیم پنجم از موتور بیرون کشیده شده است (شکل۳-۸) در این حالت به طور مشابه می‌توان با استفاده از اهم متر اتصالات را تشخیص داد. توجه کنید که در این حالت نیز مقدار مقاومت بین هر یک از سیم پیچ ها و سر مشترک نصف مقدار مقاومت بین دو سیم پیچ می‌باشد.



شکل ۳- ۸- استپر موتور ۵ سیمه

در استپر موتورهای ۴ سیمه که به صورت دو قطبی سیم بندی می شوند از ساده ترین روش اتصالات سیم بندی استپر موتور است. در این مدل استپر موتورها هم می توان با استفاده از اهم متر مقدار مقاومت بین هر یک از سیم پیچ ها را تشخیص داد.



شکل ۳- ۹- نحوه تشخیص اتصالات در استپر موتور ۴ سیمه

همچنین می‌توان با گذاشتن مولتی متر بر روی تست دیود (حالتی که در صورت وصل کردن دو پراب به هم مولتی متر بوق می‌زند) این تشخیص را داد. بدین صورت که در صورت اتصال مولتی متر به یک سیم پیچ صدای بوق شنیده خواهد شد.

۳-۱-۱-۳. تعداد فاز استپر موتور

در استپر موتورها ممکن است تعداد مختلفی سیم پیچ وجود داشته باشد. اما این سیم پیچ ها به شکل گروه هایی به نام فاز با هم در ارتباط هستند. تمامی سیم پیچ های یک فاز با یکدیگر شروع بکار می کنند. استپر موتور ها عموما دارای ۲ فاز هستند، اما استپر موتورهای ۳ فاز و ۵ فاز هم موجود هستند. عموما فازهای بیشتر قدرت استپر موتور را بیشتر خواهند کرد.

جدول ۳- ۱- مقایسه انواع استپر موتورها از نظر تعداد فاز

	استپر موتور دو فاز	استپر موتور سه فاز	استپر موتور پنج فاز
سرعت کارکرد	کم	متوسط	زیاد
دقت کارکرد	کم	متوسط	زیاد
صرفه اقتصادی	زیاد	متوسط	کم
موجودی بازار	زیاد	کم	متوسط
گام حرکت پیش فرض	۱.۸ درجه	۱.۲ درجه	۰.۷۵ درجه

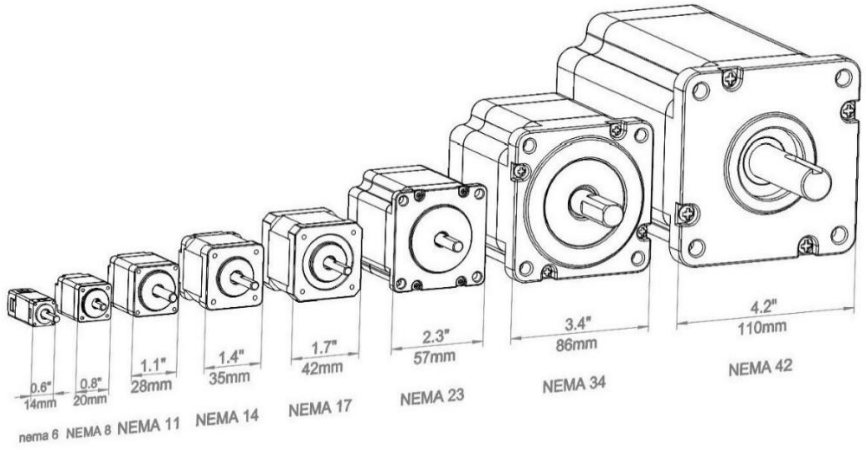
به طور معمول زمانیکه نیاز به دقت و سرعت بالا داشته باشیم، از استپ موتور ۵ فاز استفاده می‌کنیم. زیرا این نوع از موتور پله ای، با درایور معمولی، لرزش بسیار کمی ایجاد می‌کند و نیازی به درایو میکرو استپ ندارد. از سوی دیگر اگر بخواهیم حرکت‌هایی با رفت و برگشت زیاد و سریع داشته باشیم، به سراغ استپ موتور ۳ فاز می‌رویم. با این حال استپ موتور ۲ فاز با درایور معمولی لرزش زیادی را ایجاد می‌کند و نیاز به استفاده از درایوهای پیشرفته با قابلیت میکرو استپ می باشد.

۳-۱-۱-۴. سایز استپر موتور

استپر موتورها بر اساس سایز فلنج طبقه بندی می شوند بدین صورت در دو استاندارد قابل بیان هستند. اولین و پرکاربردترین استاندارد یا نماد تحت عنوان نیما یا NEMA[1] شناخته می شود که این استاندارد به ابعاد خارجی فلنج بر حسب اینچ اشاره دارد بطور مثال یک استپر موتور NEMA 17 دارای ابعاد ۱.۷ اینچ مربع می باشد یا یک استپر موتور NEMA 34 دارای ابعاد ۳.۴ اینچ مربع می باشد.

[1] National Electrical Manufacturers Association

البته این استاندارد ها بر اساس اینچ درکدگذاری محصولات جدید برندهای مختلف در حال کمتر شدن می باشد و بیشتر ترجیح می دهند که بر حسب میلی متر آنرا بیان کنند. بطور مثال برای یک استپر موتور NEMA 17 که سایز فلنج آن ۱.۷ اینچ مربع می باشد با عدد ۴۲ برحسب میلی متر بیان می کنند.



شکل ۳- ۱۰- ابعاد و اندازه های مختلف انواع استپر موتورها

۳-۱-۱-۵. گام استپر موتور

منظور از گام، کوچک ترین چرخش قابل کنترل در استپر موتور است که بر حسب زاویه بیان می شود. استپ موتورها بسته به نوع کاربرد با گام های مختلفی ساخته می شوند. بطور مثال اکثر استپر موتورها دارای گام ۱.۸ درجه می باشند بدین معنا که هر پله استپر موتور با پله دیگر به میزان ۱.۸ درجه اختلاف زاویه دارد.

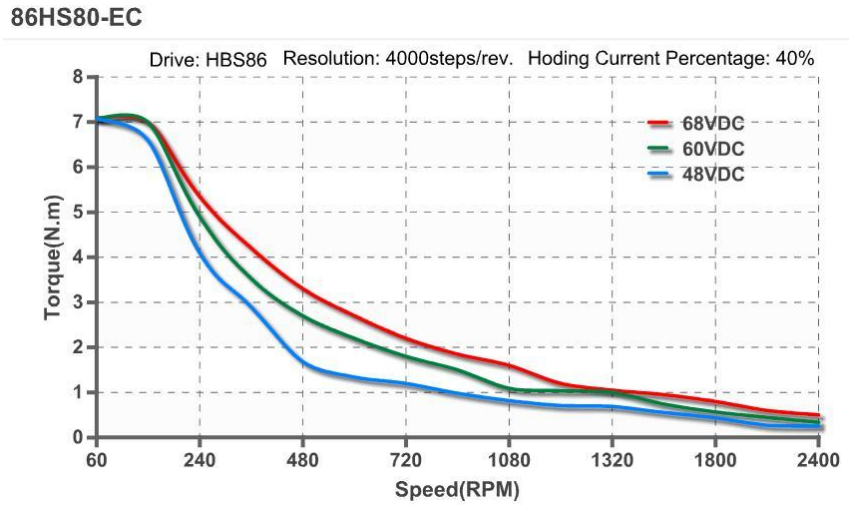
از آنجایی که استپر موتور برای چرخش یک دور کامل باید ۳۶۰ درجه طی کند می توان با تقسیم این عدد بر میزان گام استپر موتور تعداد پله ها را مشخص کرد در نتیجه می توان گفت یک استپر موتور با گام ۱.۸ درجه دارای ۲۰۰ پله می باشد.

۳-۱-۱-۶. گشتاور استپر موتور

در استپر موتور از گشتاور به عنوان گشتاور نگهداری[1] یاد می شود که به حداکثر گشتاور استپر موتور در حالت سکون اشاره دارد. این گشتاور در استپر موتور زمانی ایجاد می شود که موتور به درایو متصل است اما موتور حرکتی نمی کند. به همین دلیل مقدار این گشتاور در مقایسه با مقادیری که در زمان حرکت استپر موتور ایجاد می شود، بیشتر خواهد بود.

[1] Holding Torque

هرچه دور یا سرعت استپر موتور بیشتر شود، گشتاور نگهداری آن کمتر خواهد شد. به عبارت بهتر رابطه میان گشتاور و دور استپر موتورها بر اساس منحنی شکل ۳-۴ مشخص می شوند.



شکل ۳- ۱۱- منحنی گشتاور دور یک نمونه استپر موتور

البته روش اتصال سیم های استپر موتور در کنار ولتاژ اعمال شده بر روی آن، از متغیرهای تاثیرگذار بر روی این منحنی است. با این حال نکته ای که به وضوح با استفاده از این منحنی به دست می آید، این است که پس از سرعت ۲۰۰ تا ۳۰۰ دور، گشتاور تولید شده توسط استپر موتور به شکل قابل توجهی افت پیدا می کند به همین خاطر از استپر موتورها بیشتر در سرعت های پایین استفاده می شود.

۳-۲. تابلو فرمان ربات اسکارا

در تابلو فرمان ربات اسکارا تجهیزاتی همچون درایو استپر موتور، منبع تغذیه، ترمینال ها و کنترلر جای می گیرد که در ادامه به بررسی کامل تری از مشخصات اجزای تابلو فرمان ربات اسکارا می پردازیم. در ابتدا به تعاریف و مفاهیم اولیه این اجزا و سپس مشخصات دقیق هر یک خواهیم پرداخت.

۳-۲-۱. معرفی و بررسی درایو استپر موتور

به منظور افزایش سرعت و دقت استپر موتورها معمولاً با درایورهای مخصوصی کنترل می‌شوند که امکان تنظیم جریان و ولتاژ را فراهم می‌کنند، به این ترتیب به کاهش لرزش و افزایش گشتاور کمک می‌کنند.

شکل ۳- ۱۲- معرفی و بررسی درایو استپر موتور

درایورهای استپر موتور وظیفه دارند که پالس‌های ولتاژ را به گونه ای به فازهای استپر موتور بدهند که شفت استپر موتور حرکت کند. این پالس‌ها باید جریان کافی را برای سیم پیچ های موتور فراهم کند تا امکان حرکت استپر موتور با گشتاور کافی فراهم گردد.

از طرفی هم باید کنترل جریان فازها به نحوی صورت پذیرد که جریان موتور بیش از حد زیاد نشود که منجر به آسیب دیدن استپر موتور شود. علاوه بر این باید امکان رسیدن به حداکثر جریان در زمان کوتاه نیز برای سیم پیچ‌های موتور فراهم شود. همه این موارد بر عهده درایور استپر موتور می باشد.

درایورها در اصل بخش کنترل و یا برنامه‌ریزی تولید سیگنال کنترلی نیستند؛ بلکه واسطی برای دریافت سیگنال از کنترلر و تقویت آن در قالب پالس‌های جریانی به سمت استپر موتورها هستند. لذا اساس راه اندازی درایورهای استپ موتور در چگونگی برنامه ریزی کنترلر می باشد.

۳-۲-۱-۱. مفهوم میکرو استپ[1]

همانطور که می دانید استپر موتورها دارای یک زاویه گام پیشفرض هستند. در صورتیکه دقت حرکتی بیشتر از این مقدار نیاز باشد، می بایست از قابلیت میکرو استپ درایور که به صورت دیپ سوئیچ هایی روی درایور تعبیه شده استفاده نمود و با استفاده از کنترلر، فرمان های حرکتی را به آن منتقل کرد.

شکل ۳- ۱۳- دیپ سوئیچ های تنظیم میکرو استپ درایو استپر موتور

بنابراین میکرو استپ تکنیکی است که هر گام کامل استپر موتور را به گام‌های کوچک‌تر و دقیق‌تر تقسیم می‌کند. در این روش درایور، پالس ها را بین سیم پیچ های مختلف موتور به شکلی تقسیم می کند که استپ موتور توانایی حرکت بین استپ ها به شکل دقیق پیدا می کند.

برای مثال برای تقسیم شدن یک پله به ۱۶ پله دیگر به یکی از سیم پیچ ها ۱/۱۶ و به سیم پیچ بعدی ۱۵/۱۶ از جریان را اختصاص می دهد، در مرحله بعدی ۲/۱۶ و ۱۴/۱۶ اختصاص داده می شود و همین روند ادامه خواهد داشت. درایوی که پله های استپر موتور را به میکرو استپ تبدیل می کند در حقیقت جریان الکتریکی را در محور یک موج سینوسی افزایش و کاهش می دهد. در این روش هیچ قطبی به شکل کاملا خاموش یا روشن وجود ندارد.

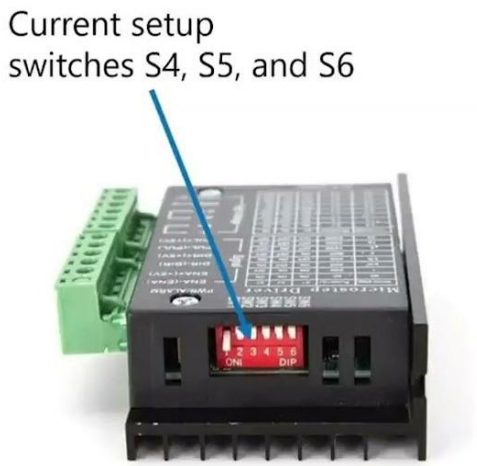
هرچه تعداد میکرو استپ ها بیشتر باشد، دقت حرکت استپر موتور هم بالاتر می‌رود و تا حد زیادی می‌توان لرزش ایجاد شده در استپ موتور را کنترل کرد. با این حال استفاده از این روش می‌تواند گشتاور موتور را کاهش دهد.

[1] Micro Stepping

۳-۲-۱-۲. تنظیم حد جریان مجاز

برای تولید گشتاور در استپر موتور، باید سیم پیچ ها جریان الکتریکی دریافت کنند، پس جریان مصرفی توسط استپر موتور ارتباط مستقیمی با نوع کارکرد و گشتاور استپ موتور خواهد داشت و یکی از عوامل مهم در تعیین قدرت استپ موتور خواهد بود.

همانند قابلیت میکرو استپ درایو برای تنظیم حد جریان استپر موتور نیز می توان از دیپ سوئیچ هایی که در کنار دیپ سوئیچ های تنظیم میکرو استپ درایو قرار دارند استفاده نمود. توجه داشته باشید که همیشه میزان حد جریان تنظیمی از حداکثر جریان قابل تحمل سیم پیچ های استپر موتور کم تر باشد.



Current(A)	PK Current	S4	S5	S6
0.5	0.7	ON	ON	ON
1.0	1.2	ON	OFF	ON
1.5	1.7	ON	ON	OFF
2.0	2.2	ON	OFF	OFF
2.5	2.7	OFF	ON	ON
2.8	2.9	OFF	OFF	ON
3.0	3.2	OFF	ON	OFF
3.5	4.0	OFF	OFF	OFF

شکل ۳- ۱۴- تنظیم حد جریان مصرفی استپر موتور

۳-۲-۱-۳. انتخاب منبع تغذیه مناسب

برای رسیدن به بهترین کارایی، انتخاب منبع تغذیه مناسب از اهمیت زیادی برخوردار است. اگر سرعت کار استپر موتور بالا باشد باید ولتاژ خروجی تا سقف مجاز بالا برود که ولتاژ بالاتر برای تغذیه، باعث لرزش موتور در سرعت های پایین شده و احتمال خرابی را برای درایو افزایش می دهد اما اگر سرعت مورد نیاز زیاد نیست بهتر است ولتاژ تغذیه پایین تر انتخاب شود که باعث کاهش نویز و حرارت و افزایش کارایی می شود

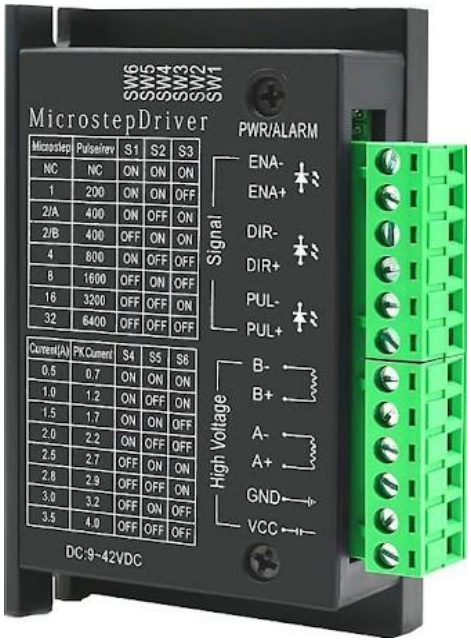
اگر بار تحمیل شده به استپر موتور زیاد باشد باید جریان خروجی منبع تغذیه بالا باشد تا گشتاور بیشتری ایجاد شود که این عوامل باعث گرم شدن زیاد موتور و درایو خواهد شد و سیستم خنک کننده مورد نیاز می باشد.

جدول ۳- ۲- ولتاژ مورد نیاز استپر موتور متناسب با سایز آن

سایز استپر موتور	ولتاژ مورد نیاز
استپر موتور NEMA 8 الی NEMA 17	12 – 24 VDC
استپر موتور NEMA 23	24 – 48 VDC
استپر موتور NEMA 34	48 – 100 VDC / 30 – 70 VAC
استپر موتور NEMA 42	110 – 220 VAC

۳-۲-۱-۴. ترمینال های سیگنال ورودی درایو

ترمینال های درایو استپر موتور در تمامی مدل ها به سه بخش اتصالات کنترلر و اتصالات موتور و اتصالات منبع تغذیه تقسیم بندی می شوند.



شکل ۳- ۱۵- ترمینال های درایو استپر موتور

- **ترمینال سیگنال ENA**

این ورودی برای فعال و غیر فعال کردن درایو استپر موتور کاربر دارد. در صورت دریافت ولتاژ توسط این ورودی، استپر موتور آزاد خواهد شد و چراغ سبز روی درایو چشمک خواهد زد و در صورت آزاد بود این ترمینال، استپر موتور زیر بار خواهد رفت و چراغ سبز ثابت روشن خواهد بود.

سطح صفر پالس بین ۰ الی ۰.۵ ولت بوده و سطح یک پالس نیز بین ۴ الی ۵ ولت می باشد. اگر ولتاژ ورودی بالاتر از ۵ ولت باشد باید یک مقاومت محدود کننده جریان بصورت سری با ورودی قرار گیرد.

- **ترمینال سیگنال DIR**

این ورودی برای تعیین و تغییر جهت چرخش استپر موتور کاربرد دارد، برای تغییر جهت چرخش باید این ترمینال یک ولتاژ در محدوده ۴ الی ۵ ولت دریافت کند که البته جهت مثبت و منفی ولتاژ ورودی باید حتما رعایت شود. اگر ولتاژ ورودی بالاتر از ۵ ولت باشد باید یک مقاومت محدود کننده جریان بصورت سری با ورودی قرار گیرد.

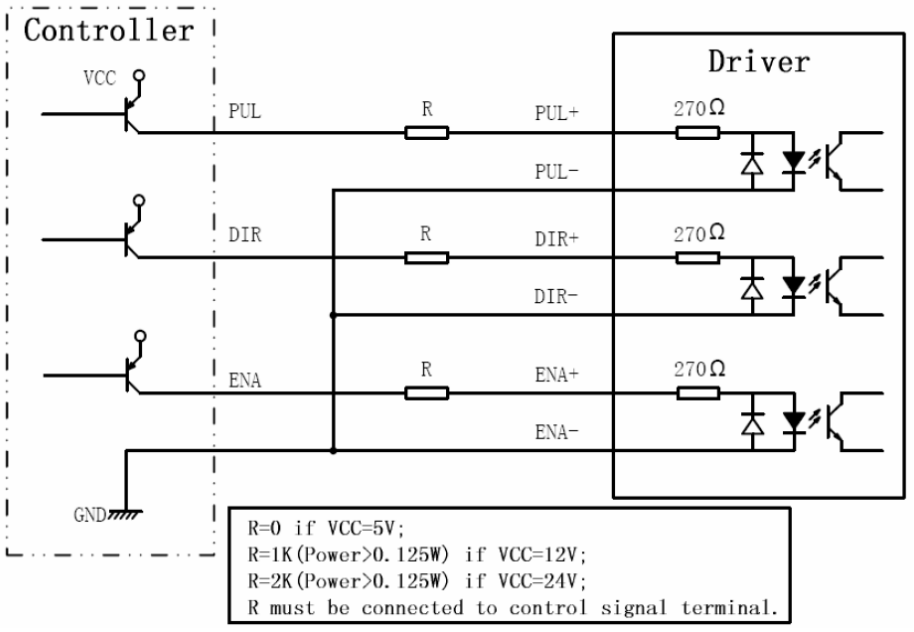
- **ترمینال سیگنال PUL**

این ورودی برای ارسال تعداد پالس لازم از طرف کنترلر برای چرخش استپر موتور می باشد، که با توجه به سرعت و دقت مورد نیاز فرکانس پالس ورودی و تنظیم میکرو استپ محاسبه می شود.

سطح صفر پالس بین ۰ الی ۰.۵ ولت بوده و سطح یک پالس نیز بین ۴ الی ۵ ولت می باشد. اگر ولتاژ ورودی بالاتر از ۵ ولت باشد باید یک مقاومت محدود کننده جریان بصورت سری با ورودی قرار گیرد. همچنین برای عملکرد بهتر، پهنای پالس ورودی از ۱.۵ میکروثانیه کمتر نباشد.

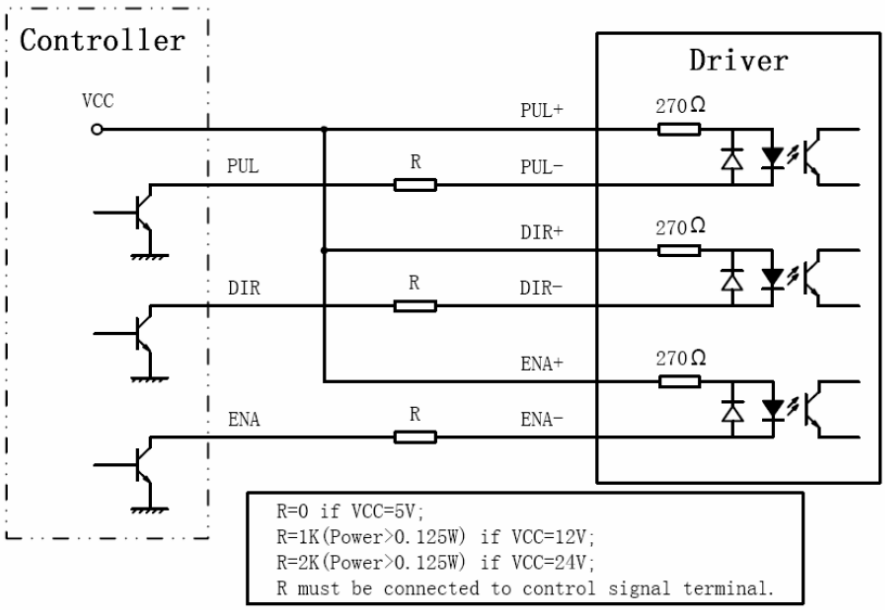
۳-۲-۱-۵. انواع اتصالات کنترلر به درایو

بطور کلی دو نوع اتصال بین کنترلر و درایو استپر موتور وجود دارد یکی اتصال PNP و دیگری اتصال NPN ، حال تفاوتی که بین این دو نوع اتصال وجود دارد به این شکل هست که در اتصال PNP پایه های منفی ترمینال های سیگنال درایو به یکدیگر متصل می شوند و توسط یک سیم مشترک به زمین کنترلر متصل می شود و به پایه های مثبت ترمینال های سیگنال درایو سطح ولتاژ ۵ ولت اعمال می شود.



شکل ۳- ۱۶- اتصال PNP یا کاتد مشترک

در اتصال NPN پایه های مثبت ترمینال های سیگنال درایو به یکدیگر متصل می شوند و توسط یک سیم مشترک به ۵ ولت کنترلر متصل می شود و به پایه های منفی ترمینال های سیگنال درایو سطح ولتاژ ۰ ولت یا زمین اعمال می شود.

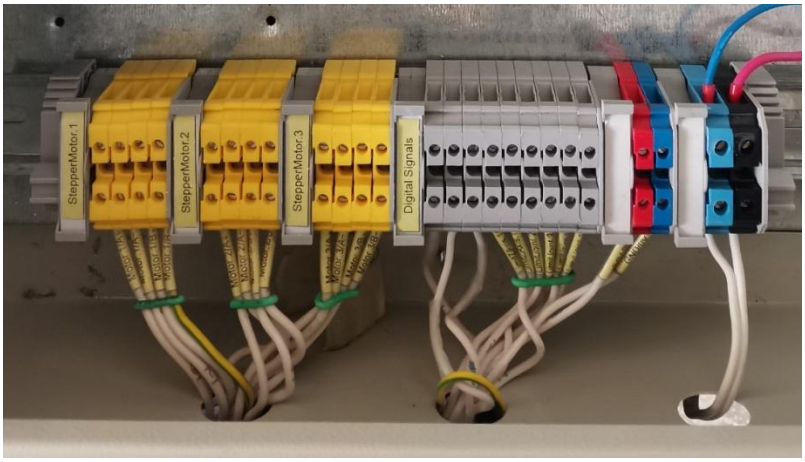


شکل ۳- ۱۷- اتصال NPN یا آند مشترک

۳-۳. اتصالات بین تابلو فرمان و ربات اسکارا

اتصالات بین تابلو فرمان و ربات اسکارا به سه بخش اتصالات استپر موتور، سنسورها و تغذیه تقسیم می شود. همانطورکه در شکل ۳-۱۸ مشخص است ترمینال های زرد به عنوان اتصالات استپر موتور به سه بخش تقسیم می شود ک هر بخش مربوط به فازهای استپر موتور می باشد.

از آنجایی که استپر موتورهای ربات اسکارا از نوع دو فاز می باشد بنابراین به چهار سیم برای اتصال هر استپر موتور نیاز خواهیم داشت و در مجموع ۱۲ ترمینال به این عنوان اختصاص داده شده است.



شکل ۳- ۱۸- ترمینال های سیگنال و قدرت تابلو فرمان ربات اسکارا

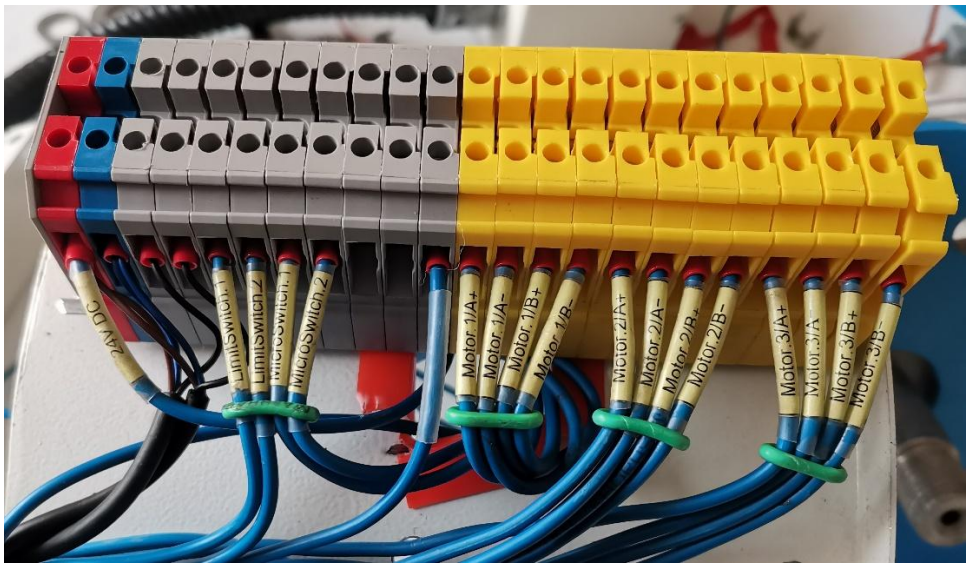
گروه دیگر از ترمینال ها تحت عنوان ترمینال های سیگنال برای اتصالات سنسورهای ربات اسکارا تعبیه شده است. بر روی هر محور ربات اسکارا دو سنسور تعبیه شده است که هر سنسور دارای یک خروجی سیگنال ۲۴ ولت می باشد و در مجموع ۶ خروجی سیگنال ۲۴ ولت از سنسورهای ربات اسکارا به ترمینال های تابلو فرمان متصل شده است.

در شکل ۳-۱۸ ترمینال های خاکستری به همین عنوان اختصاص داده شده اند که در مجموع ۹ ترمینال تعبیه شده و ۳ عدد آن به عنوان ترمینال رزرو در نظر گرفته شده است. همچنین در سمت مکانیک ربات اسکارا نیز این ترمینال ها به همین ترتیب تعبیه شده است.

در انتها نیز ترمینال های تغذیه قرار داردند که به این ترتیب به دو بخش تغذیه DC و AC تقسیم بندی می شوند. بخش تغذیه DC با دو ترمینال قرمز و آبی مشخص شده اند که به عنوان ترمینال های ۲۴ ولت در نظر گرفته شده اند و برای مصارف داخلی نظیر تغذیه سنسورها از این ترمینال استفاده می شود. بخش AC نیز تغذیه اصلی تابلو می باشد که با دو ترمینال آبی و مشکی در شکل ۳-۱۸ مشخص شده اند که به عنوان ترمینال های فاز و نول در نظر گرفته شده اند.

در بخش مکانیک ربات اسکارا نیز به همین ترتیب ترمینال های سیگنال، قدرت و تغذیه که در تابلو فرمان تعبیه شده است در این قسمت نیز در نظر گرفته شده است.

شکل ۳- ۱۹- ترمینال های سیگنال، قدرت و تغذیه ربات اسکارا



شکل ۳- ۲۰- اتصالات ترمینال های تعبیه شده بر روی ربات اسکارا

فصل چهارم: معرفی و بررسی کنترلر ربات اسکارا

۴-۱. معرفی کنترلر ربات اسکارا

در این پایان‌نامه، به طراحی و پیاده‌سازی یک کنترلر پیشرفته برای ربات اسکارا پرداخته شده است. ربات‌های اسکارا به دلیل دقت و سرعت بالا در انجام وظایف صنعتی، به یکی از پرکاربردترین ربات‌ها در صنایع مختلف تبدیل شده‌اند. هدف اصلی این پروژه طراحی و پیاده‌سازی یک کنترلر پیشرفته برای ربات اسکارا می باشد که باعث افزایش دقت، سرعت، پایداری عملکرد و کارایی ربات در انجام وظایف صنعتی خواهد شد.

کنترلر طراحی شده از یک پردازنده اختصاصی STM32 استفاده می کند این پردازنده به عنوان هسته کنترلی ربات، وظیفه کنترل حرکت و مسیریابی خودکار بصورت خطی و زاویه ای بین موقعیت ها را داراست. همچنین این کنترلر دارای یک نمایشگر برای نمایش اطلاعات کنترلی به صورت محلی[1] و یک وب سرور که امکان کنترل ربات از راه دور[2] را فراهم می‌کند.

شکل ۴- ۱- نمونه اولیه کنترلر ربات اسکارا

کنترلر طراحی شده برای ربات اسکارا یک سیستم الکترونیکی پیشرفته است که وظیفه دارد حرکات دقیق و سریع ربات را کنترل و هماهنگ کند. این کنترلر با استفاده از الگوریتم‌های پیشرفته و تکنیک‌های بهینه‌سازی، توانایی دارد به طور موثر به تغییرات محیطی پاسخ دهد و وظایف تکراری صنعتی را با دقت بالا اجرا کند. هدف اصلی این کنترلر افزایش کارایی، دقت، و پایداری ربات اسکارا در فرآیندهای تولیدی و مونتاژی است.

[1] Local
[2] Remote

۴-۲. اجزای کنترلر ربات اسکارا

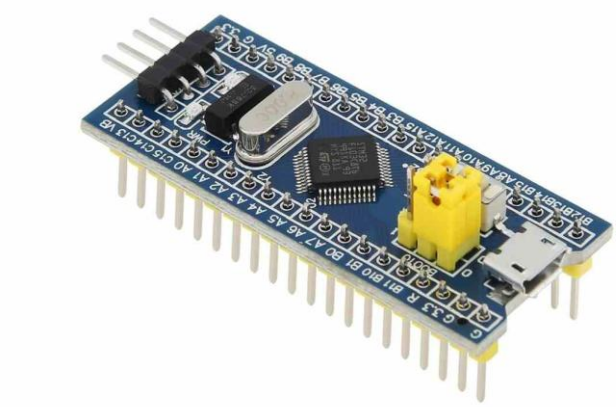
کنترلر طراحی شده از اجزای مختلفی تشکیل شده است. بطور کلی بخش های اصلی کنترلر شامل موارد زیر می باشد که در ادامه به بررسی کامل تر هر یک خواهیم پرداخت.

- میکروکنترلر STM32
- ال سی دی کاراکتری و رابط I2C
- ماژول وای فای ESP8266
- اپتوکوپلر PC817
- آی سی لاین درایو AM26LS31
- روتاری انکودر ولومی

۴-۲-۱. معرفی میکروکنترلر STM32

خانواده STM32 میکروکنترلرهای ۳۲ بیتی شرکت ST هستند. این محصول با ترکیب عملکرد بسیار قوی، ولتاژ کاری پایین، پردازش سیگنال دیجیتال و سهولت در توسعه، محبوبیت بسیار زیادی را کسب کرده است.

میکروکنترلرهای STM32 از پروتکل های ارتباطی گسترده‌ای پشتیبانی می‌کنند و همچنین میکروکنترلرهای خانواده STM32 شامل زیر مجموعه های مختلفی هستند و به دلیل زیاد بودن تعداد پردانده های این خانواده، فقط میکروکنترلر STM32F103C8T6 را معرفی میکنیم.



شکل ۴- ۲- برد توسعه میکروکنترلر STM32F103C8T6

سری STM32F1 جز پردازنده های میان رده این خانواده است، اما این سری دارای قدرت پردازشی بسیار بالایی می باشد. سری STM32F1 اولین گروه میکروکنترلرهای STM32 مبتنی بر هسته ARM Cortex-M3 است.

شکل ۴- ۳- پردازنده برد توسعه STM32F103C8T6

سری F1 با افزایش سرعت پردازنده، اندازه حافظه داخلی، انواع لوازم جانبی با گذشت زمان تکامل یافته است. هرکدام از زیر مجموعه های خانواده STM32 شامل زیر مجموعه های بسیاری می‌شوند. از محبوب ترین زیر مجموعه این خانواده، پردازنده های سری STM32F103C8T6 می باشد. از ویژگی های این پردازنده می توان به موارد زیر اشاره نمود:

- حداکثر فرکانس کاری ۷۲ مگاهرتز
- ۶۴ کیلوبایت حافظه Flash و ۲۰ کیلوبایت حافظه SRAM
- پشتیبانی از ۷ کانال DMA Controller
- پشتیبانی ازپروتکل های ارتباطی SPI و I2C و USART
- پشتیبانی از ۴ تایمر خارجی
- دارای ۳۷ پورت GPIO

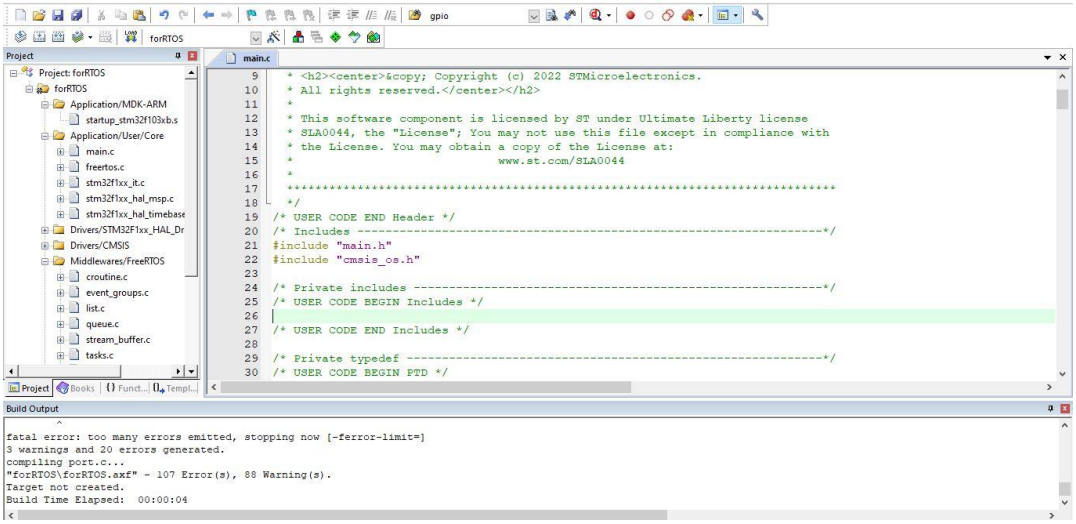
۴-۲-۱-۱. معرفی محیط های برنامه نویسی STM32

محیط های مختلفی برای برنامه نویسی میکروکنترلرهای STM32 وجود دارد. Keil MDK و STM32CubeIDE از محبوب ترین محیط های برنامه نویسی این خانواده از میکروکنترلرها هستند.

- **محیط برنامه نویسی Keil MDK**

نرم افزار Keil MDK یکی از قدرتمندترین محیط های برنامه نویسی برای ویرایش و کامپایل کد است. امکانات جانبی این نرم‌افزار ازجمله محیط دیباگ، آنالیز زمانی عملکرد میکروکنترلر، ارتباط با پروگرامرهای مختلف، انتخاب کامپایلر و غیره شما را از هر نرم‌افزار دیگری بی‌نیاز می‌کند.

این محیط برنامه نویسی نیز به عنوان کامپایلر نیز شناخته می شود چراکه از کامپایلر اختصاصی خود یعنی Keil بهره می برد همچنین لازم به ذکر است که این محیط برنامه نویسی برای پیکربندی سخت افزار نیاز به یک نرم افزار دیگر تحت عنوان STM32CubeMX دارد که باید آنرا نصب داشته باشد.

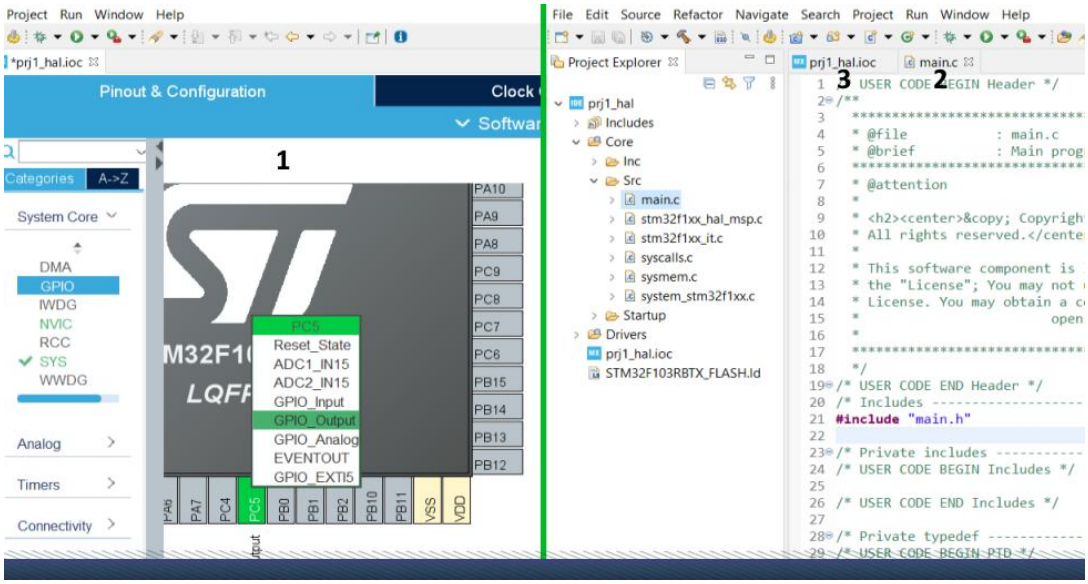


شکل ۴- ۴- محیط برنامه نویسی Keil

➢ محیط برنامه نویسی STM32CubeIDE

STM32CubeIDE در واقع یک مجموعه نرم‌افزاری می‌باشد، که همه‌ی نرم‌افزارهایی که ما برای کار با میکروکنترلرهای ST نیاز داریم، از جمله نرم افزار پیکربندی STM32CubeMX را در یک مجموعه گنجانده است و شما دیگر نیاز نیست با صرف وقت زیاد به نصب نرم‌افزارهای مختلف بپردازید.

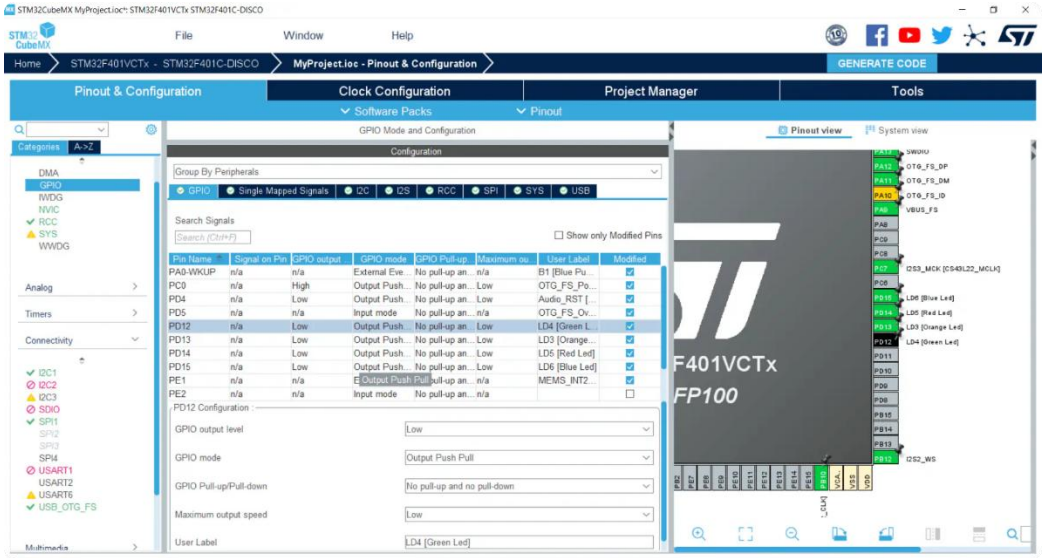
این نرم‌افزار بر اساس فریمورک Eclipse، و کامپایلر GCC برای توسعه و دیباگر (GNU Debugger GDB) برای دیباگ کردن می‌باشد.



شکل ۴- ۵- محیط برنامه نویسی STM32CubeIDE

➢ محیط پیکربندی STM32CubeMX

این نرم‌افزار از سمت شرکت ST برای پیکره‌بندی اولیه کد در نظر گرفته‌شده است. نرم‌افزار STM32CubeMX یک محیط گرافیکی بسیار زیبا و کاربرپسند دارد که فقط با استفاده از چند کلیک می‌توانید میکروکنترلر خود را پیکره‌بندی اولیه کنید.



شکل ۴- ۶- محیط پیکربندی میکروکنترلر STM32

۴-۲-۱-۲. معرفی پروگرامر STM32

معمولا میکروکنترلرهای ST را به دو روش JTAG و SWD پروگرام می‌کنند. در روش SWD نیاز به پروگرامر ST-Link و در روش JTAG نیاز به پروگرامر J-Link داریم. البته با پروگرامر J-Link به روش SWD هم می‌توانیم عمل پروگرام را انجام بدهیم.

درواقع تفاوت اصلی در این دو روش تعداد سیم‌هایی است که برای پروگرام کردن از آن استفاده می‌کنند. پروتکل SWD به تعداد سیم‌های کمتری برای عمل پروگرام نیاز دارد.

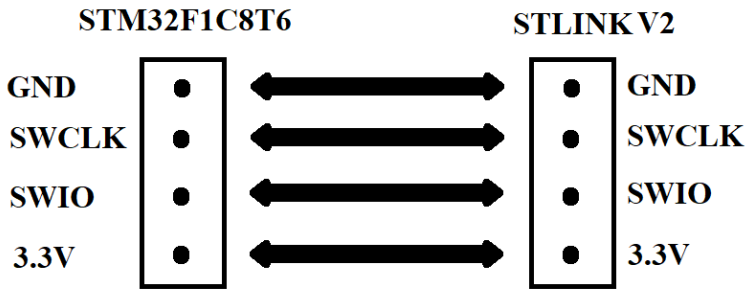
➢ روش JTAG

این روش که مخفف Joint Test Action Group است در واقع یک روش استاندارد در بین همه میکروکنترلرهای سری ARM است که برای برنامه ریزی و اشکال زدایی به وسیله دیباگر یا پروگرامر JLINK استفاده می شود.

➢ روش SWD

در روش SWD ما فقط از دو پایه به اسم های SWDIO و SWCLK استفاده می کنیم که در واقع این پایه ها در پروتکل JTAG هم وجود دارند. در این روش از پروگرمرهای ST-LINK شرکت ST استفاده می کنیم که در مجموع ۴ پایه استفاده می شود.

این روش چون از پایه های کمتری استفاده می کند مرسوم تر و محبوب تر است. در شکل ۴-۷ نحوه اتصال میکروکنترلر به پروگرامر STLINK مشخص شده است.



شکل ۴- ۷- نحوه اتصال پروگرامر به میکروکنترلر

۴-۲-۲. معرفی LCD کاراکتری و رابط I2C

ال سی دی کاراکتری، نمایشگرهای ساده‌ای هستند که برای نمایش حروف، اعداد و نمادهای ساده طراحی شده‌اند. این نمایشگرها به دلیل سادگی، قیمت مناسب و مصرف کم انرژی، در بسیاری از دستگاه‌های الکترونیکی روزمره مورد استفاده قرار می‌گیرند.

شکل ۴- ۸- معرفی LCD کاراکتری و رابط I2C

۴-۲-۲-۱. انواع ال سی دی کاراکتری از نظر سطر و ستون

ال سی دی ۲×۱۶: رایج‌ترین نوع ال سی دی کاراکتری، ال سی دی ۲×۱۶ است که ۲ سطر و ۱۶ ستون دارد. این نوع ال سی دی برای نمایش اطلاعات ساده و کوتاه بسیار مناسب است.

ال سی دی ۴×۲۰: این نوع ال سی دی دارای ۴ سطر و ۲۰ ستون است و برای نمایش اطلاعات بیشتر نسبت به ال سی دی ۲×۱۶ مناسب است.

۴-۲-۲-۲. معرفی پایه های LCD کاراکتری

این نوع ال سی دی ها بسته به اینکه دارای نور پشت زمینه باشند یا نه دارای ۱۴ یا ۱۶ پایه هستند که در جدول زیر این پایه ها معرفی شده اند.

جدول ۴- ۱- معرفی پایه های LCD کاراکتری

پایه	نماد	توضیح
1	VSS	زمین منبع تغذیه
2	VDD	ولتاژ ۵ ولت منبع تغذیه
3	VEE	ولتاژ کنترل شدت نور صفحه
4	RS	اگر RS = 0 باشد رجیستر دستور انتخاب می شود. اگر RS = 1 باشد رجیستر داده انتخاب می شود.
5	R/W	R/W = 0 برای نوشتن اطلاعات R/W = 1 برای خواندن اطلاعات
6	E	فعال ساز LCD
7	D0	بیت ۰ باس داده
8	D1	بیت ۱ باس داده
9	D2	بیت ۲ باس داده
10	D3	بیت ۳ باس داده
11	D4	بیت ۴ باس داده
12	D5	بیت ۵ باس داده
13	D6	بیت ۶ باس داده
14	D7	بیت ۷ باس داده
15	BLA	آنود LED پشت زمینه LCD
16	BLK	کاتود LED پشت زمینه LCD

- **پایه RS**

در داخل ال سی دی دو ریجستر وجود دارد که توسط پایه RS انتخاب می شوند. اگر این پایه صفر شود، ریجستر دستور[1] انتخاب می شود تا فرمان هایی مانند پیکربندی LCD ، پاک کردن آن، جابجایی مکان نما و ... برای ال سی دی ارسال می شود. در صورتی که اگر این پایه یک شود، ریجستر داده[2] انتخاب می شود تا کاربر بتواند اطلاعاتی را که می خواهد روی ال سی دی بنویسد، برای ال سی دی ارسال کند.

- **پایه R/W**

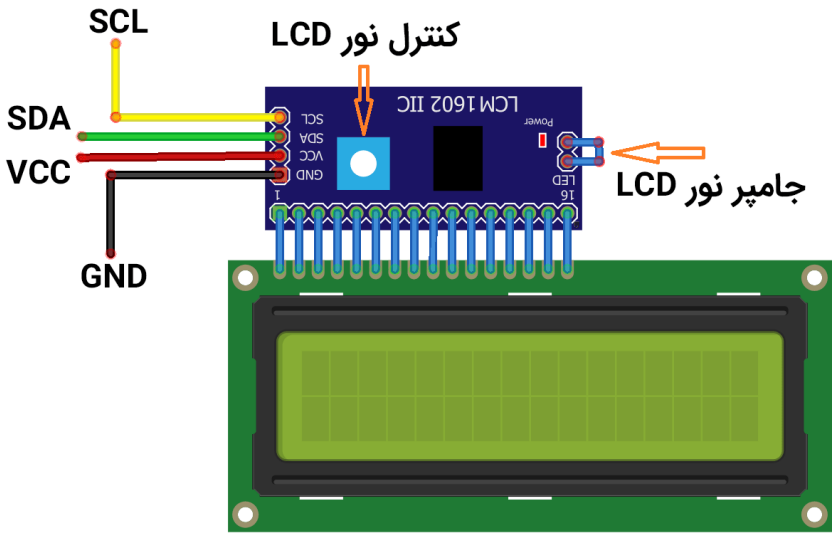
به کمک این پایه، کاربر مشخص می کند که می خواهد اطلاعات را روی ال سی دی بنویسد یا از روی آن بخواند. اگر این پایه یک شود، اطلاعات از روی ال سی دی خوانده می شود و در صورتی که صفر شود، اطلاعات روی آن نوشته می شود.

- **پایه E**

اگر در این پایه، پالسی از یک به صفر قرار داده شود، اطلاعاتی که روی پایه های D0 تا D7 قرار دارد درون یکی از ریجستر هایی که توسط پایه RS مشخص می شود جای می گیرد. حداقل زمانی که این پایه باید صفر باشد، ۴۵۰ نانوثانیه است.

۴-۲-۲-۳. نحوه اتصال LCD به رابط I2C

برای اتصال ال سی دی به میکروکنترلر بدلیل اینکه از تعداد پایه های کمتری استفاده نماییم از رابط I2C برای این نوع ال سی دی استفاده می نماییم.



شکل ۴- ۹- نحوه اتصال LCD به رابط I2C

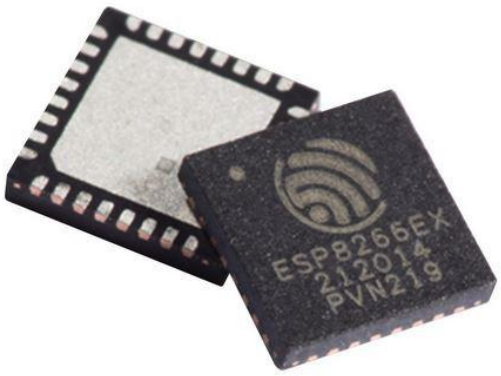
[1] Instruction Register
[2] Data Register

۴-۲-۳. معرفی ماژول وای فای ESP8266

ماژول ESP8266 در حقیقت نوعی میکروکنترلر با قابلیت برنامه نویسی مستقل است که توسط شرکت Espressif Systems توسعه داده شده است. این ماژول امکان اتصال دستگاه‌ها به شبکه‌های بی‌سیم را فراهم می‌کند و از معماری کوچک و قدرتمند خود برای ایجاد برنامه‌های قابل اجرا بر روی خود استفاده می‌کند.

شکل ۴- ۱۰- معرفی ماژول وای فای ESP8266

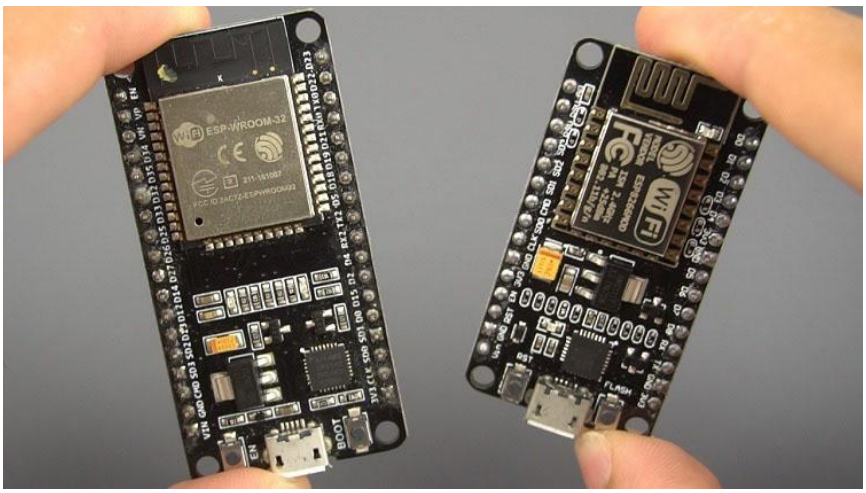
ماژول ESP8266 دارای یک پردازنده Tensilica Xtensa LX106 با فرکانس ۸۰ مگاهرتز است که با حافظه فلش داخلی ۴ مگابایت قابل برنامه‌ریزی است. علاوه بر این، این برد دارای چیپ Wi-Fi با استاندارد 802.11 b/g/n است که به دستگاه‌ها امکان اتصال به شبکه‌های بی‌سیم را می‌دهد.



شکل ۴- ۱۱- تراشه ماژول ESP8266

ماژول ESP8266 دارای پورت‌های UART، I2C و SPI است که به کاربر امکان اتصال به سنسورها و ماژول‌های دیگر را می‌دهد که این قابلیت‌ها، تعامل با دستگاه‌های خارجی را برای پروژه‌های مختلف بسیار آسان می‌کند. این برد از پروتکل‌های رمزنگاری WPA/WPA2 و WEP برای اتصال امن به شبکه‌های بی‌سیم استفاده می‌کند و قابلیت احراز هویت و رمزگذاری را فراهم می‌کند.

تراشه‌های خانواده ESP در بردهای مختلفی استفاده شده‌اند و امروزه کاربرد بسیار گسترده‌ای را پیدا کرده‌اند. برد NodeMCU یکی از پرمصرف ترین بردهای امبدد در زمینه اینترنت اشیاء[1] هستند. هسته NodeMCU دارای دو نسخه‌ی مختلف است. یکی دارای چیپ ESP8266 ورژن ESP12 و دیگری دارای چیپ ESP32 است.



شکل ۴- ۱۲- برد NodeMCU ماژول ESP8266 و ESP32

۴-۲-۳-۱. تفاوت ماژول های ESP8266 و ESP32

ماژول ESP8266 یک میکروکنترلر با قابلیت Wi-Fi است و ماژول ESP32 علاوه بر قابلیت Wi-Fi دارای قابلیت اتصال بلوتوث نیز است. لذا تفاوت اولیه بین این ماژول ها در نوع میکروکنترلر استفاده شده در آنها است. ماژول ESP32 نسبت به ESP8266 قدرت پردازش بیشتری دارد. بدین معنا که ESP32 قادر به اجرای عملیات پیچیده‌تر، محاسبات متقدم و پردازش داده‌های بیشتر است.

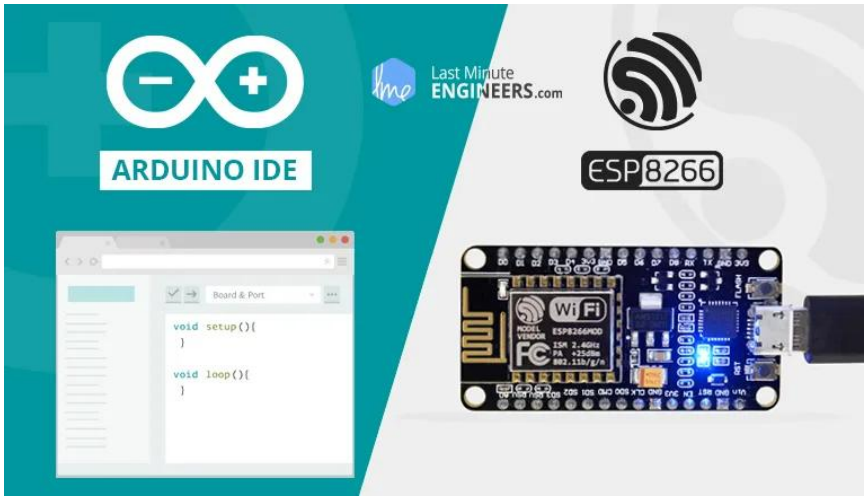
ماژول ESP32 دارای حافظه فلش و حافظه SRAM بیشتری نسبت به ESP8266 است. بدین معنا که ESP32 قادر به ذخیره و پردازش حجم بیشتری از داده‌ها است. ESP32 دارای تعداد بیشتری پورت و واحد ارتباطی است که امکان اتصال به سایر ماژول ها را می‌دهد.

ماژول ESP32 دارای قابلیت بلوتوث است که به شما امکان ارتباط با دستگاه‌های بلوتوثی را می‌دهد. این ویژگی برای کاربردهایی مانند اتصال به دستگاه‌های پرینتر بلوتوث، سنسورهای بلوتوث و دیگر دستگاه‌های هوشمند کاربرد دارد. هر دو ماژول ESP8266 و ESP32 دارای قابلیت اتصال به شبکه Wi-Fi هستند. اما ESP32 از ویژگی‌های پیشرفته‌تری مانند حالت‌های آنتن داخلی و خارجی و پشتیبانی از مد روتری پیشرفته (WDS) برخوردار است.

[1] Internet of Things (IoT)

۴-۲-۳-۲. زبان برنامه نویسی ماژول ESP8266

ماژول ESP8266 با استفاده از زبان برنامه‌نویسی C++ و با استفاده از ابزار توسعه Arduino IDE برنامه‌ریزی می‌شود. این زبان برنامه‌نویسی بر اساس زبان C است و به توسعه‌دهندگان اجازه می‌دهد کدهای کامپایل شده را بر روی برد ESP8266 اجرا کنند.



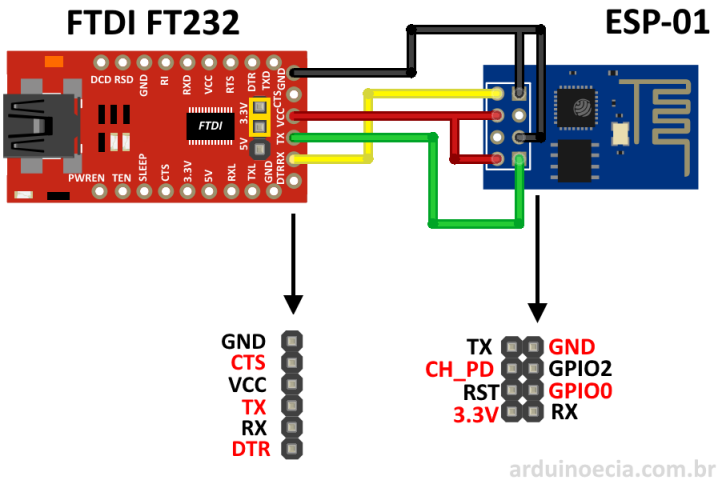
شکل ۴- ۱۳- برنامه نویسی ماژول ESP8266 توسط نرم افزار Arduino IDE

ماژول ESP8266 علاوه بر زبان C++، دارای قابلیت برنامه‌ریزی با استفاده از زبان‌های برنامه‌نویسی دیگری نظیر MicroPython و Lua نیز است. اما برای برنامه‌ریزی اصلی و پرکاربرد ESP8266، استفاده از زبان C++ و Arduino IDE توصیه می‌شود.

با استفاده از Arduino IDE و زبان C++ می‌توانید کدهای برنامه‌ریزی را برای کنترل دستگاه‌ها، ارتباط با شبکه Wi-Fi، خواندن و نوشتن داده‌ها، کنترل پین‌ها و توابع دیگر بر روی ESP8266 نوشته و اجرا کنید. همچنین، بسته به نیاز پروژه، می‌توانید از کتابخانه‌های آماده مربوط به ESP8266 استفاده کنید تا توسعه را سریع تر و آسان‌تر کنید.

۴-۲-۳-۳. نحوه پروگرام کردن ماژول ESP8266

بردها و تراشه‌های ESP توسط ارتباط سریال یا همان UART قابلیت اتصال به کامپیوتر را دارند. بنابراین برای پروگرام کردن مستقیم آن‌ها بایستی از یک مبدل USB به سریال استفاده نمود. در مدل‌هایی از ESP که صرفا تراشه نیستند و روی برد خاصی قرار گرفته‌اند از جمله بردهای NodeMCU این قابلیت در برد فراهم شده است. کافیست از درگاه USB روی برد استفاده کرده و به کامپیوتر اتصال دهید.

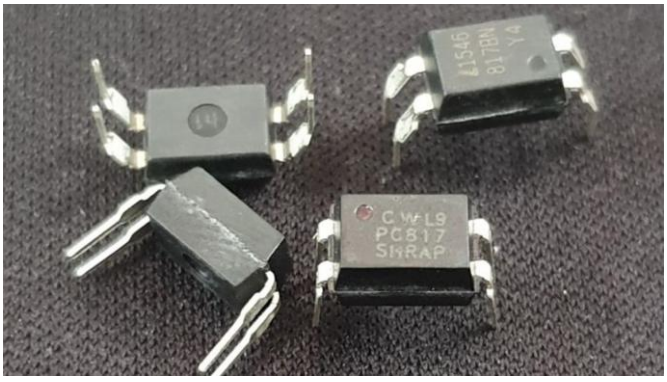


شکل ۴- ۱۴- نحوه اتصال مستقیم ماژول ESP8266 به مبدل USB به سریال

برای پروگرام کردن ماژول ESP8266 ابتدا مانند شکل ۴-۱۴ اتصالات بین ماژول و مبدل USB به سریال را برقرار کنید و سپس مبدل را به کامپیوتر خود متصل کنید. در مرحله بعد پایه GPIO0 ماژول را به زمین متصل کرده و سپس پایه RESET ماژول را برای لحظه ای زمین کنید. درنهایت هر دو پایه GPIO0 و RESET را به حالت اولیه برگردانید (از حالت زمین خارج کنید). در این حالت ماژول ESP8266 به حالت بوت یا پروگرام شدن وارد می‌شود و به راحتی با انجام مراحل و اتصالات بالا از طریق نرم افزار آردوینو می توانید ماژول را پروگرام نمایید.

۴-۲-۴. معرفی اپتوکوپلر PC817

اپتوکوپلر[1] یا اپتوایزولاتور[2] یک قطعه الکترونیکی است که با استفاده از نور، سیگنال های الکتریکی را بین دو مدار مجزا انتقال می دهد و برای جدا کردن و محافظت از مدار در برابر آسیب های الکتریکی یا نویزهای ناخواسته به ویژه در مدارهای با ولتاژ پایین و مدارهای حساس به نویز استفاده می شود.



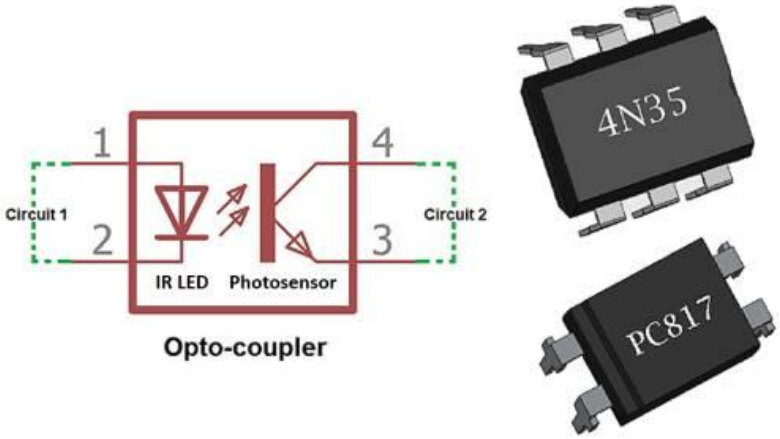
شکل ۴- ۱۵- معرفی اپتوکوپلر PC817

[1] Optocoupler
[2] Optoisolator

۴-۲-۴-۱. نحوه کار اپتوکوپلر PC817

همانطورکه در شماتیک مربوط به اپتوکوپلر PC817 در شکل ۴-۱۶ مشخص است، پین های ۱ و ۲ مربوط به LED هستند. LED نور مادون قرمز را به سمت یک گیرنده نوری (فوتوترانزیستور) ساطع می کند و فوتوترانزیستور نور ساطع شده از LED را دریافت می کند. فوتوترانزیستور شبیه یک ترانزیستور BJT عمل کرده و مدار خروجی را با استفاده از پایه های کلکتور[1] و امیتر[2] سوئیچ می کند.

فضای بین فوتوترانزیستور و LED، از جنس یک ماده شفاف و غیر رسانا ساخته شده که در واقع نارسانای الکتریکی ولی رسانای نوری است مثل شیشه، هوا یا پلاستیک شفاف. این ماده دو مدار مختلف را از نظر الکتریکی از هم جدا و ایزوله می کند. ایزولاسیون الکتریکی به طور معمول در محدوده ۱۰ کیلو ولت یا بالاتر است.

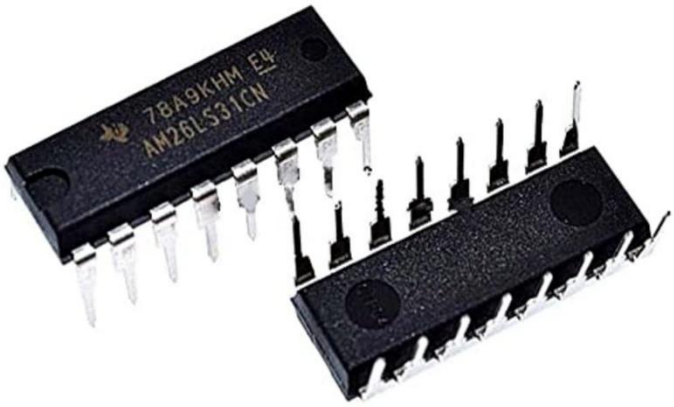


شکل ۴- ۱۶- نحوه کار اپتوکوپلر PC817

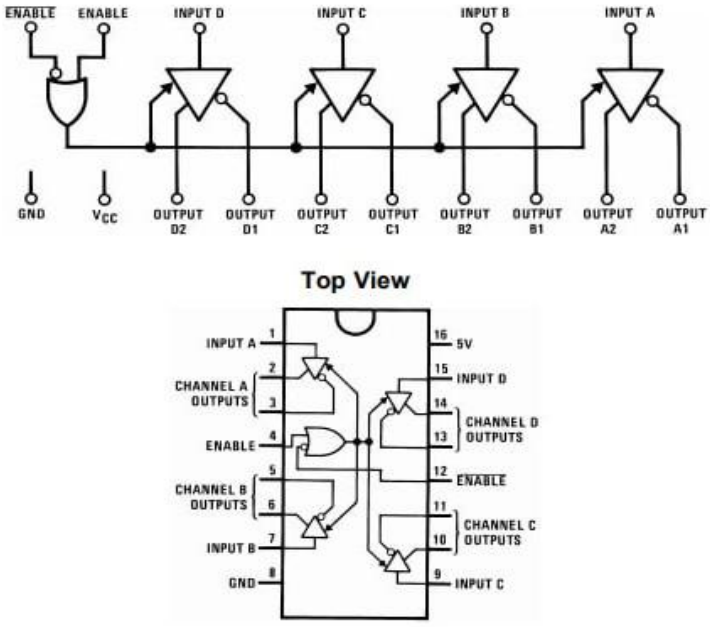
[1] Collector
[2] Emitter

۴-۲-۵. معرفی آی سی لاین درایو AM26LS31

لاین درایو[1] نوعی مدار الکترونیکی است که برای تقویت سیگنال‌های ورودی و انتقال آن‌ها به فاصله‌های طولانی یا برای تطبیق سیگنال‌های مختلف با دستگاه‌های ورودی و خروجی دیگر استفاده می‌شوند. این نوع درایورها معمولاً به صورت آی سی‌های خاصی طراحی می‌شوند که قادر به تقویت و اصلاح سیگنال‌ها هستند تا اطمینان حاصل شود که سیگنال‌ها به درستی و بدون کاهش کیفیت منتقل می‌شوند.



شکل ۴- ۱۷- آی سی لاین درایو AM26LS31



شکل ۴- ۱۸- شماتیک آی سی لاین درایو AM26LS31

[1] Line Drive

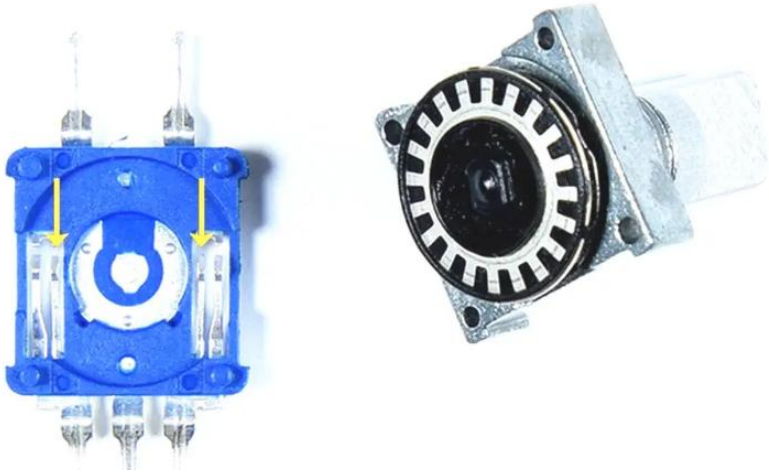
۴-۲-۶. معرفی روتاری انکودر ولومی

روتاری انکودر ولومی نوعی سنسور موقعیت است که موقعیت زاویه شفت را به دیتای دیجیتال تبدیل می‌کند. این انکودر از نوع افزاینده بوده که ساده‌ترین روش برای اندازه‌گیری موقعیت زاویه‌ای به شمار می‌رود. این انکودر همچنین دارای یک سوئیچ است که فشردن شفت را تشخیص می‌دهد.

شکل ۴- ۱۹- یک نمونه روتاری انکودر ولومی

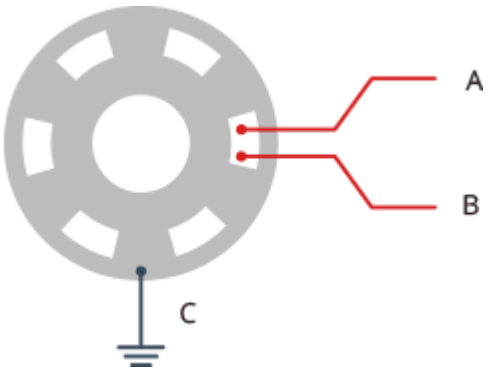
۴-۲-۶-۱. نحوه کار روتاری انکودر ولومی

در داخل روتاری انکودر یک عدد دیسک شیار دار وجود دارد که این دیسک رسانا است و به پایه GND روتاری انکودر متصل می شود. همچنین دو عدد پایه بین شیارها قرار دارند که در حال عادی با دیسک برخوردی ندارند. در شکل ۴-۲۰ ساختار داخلی یک روتاری انکودر را مشاهده می کنید.



شکل ۴- ۲۰- ساختار داخلی یک روتاری انکودر ولومی

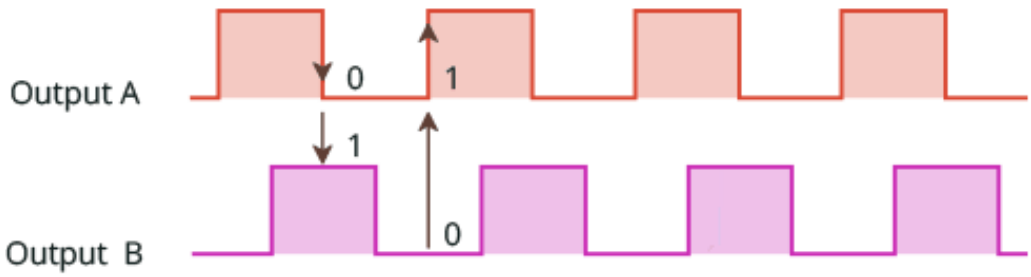
همانطور که در بالا گفته شد دو عدد پایه با نام های A و B وجود دارند که با چرخش انکودر یکی پس از دیگری با دیسک تماس پیدا کرده و یک لحظه سطح ولتاژ آنها صفر می شود که بر اساس همین ترتیب اتصال پایه های A و B به زمین، جهت چرخش را تشخیص می دهیم.



شکل ۴- ۲۱- دیسک شیار دار روتاری انکودر

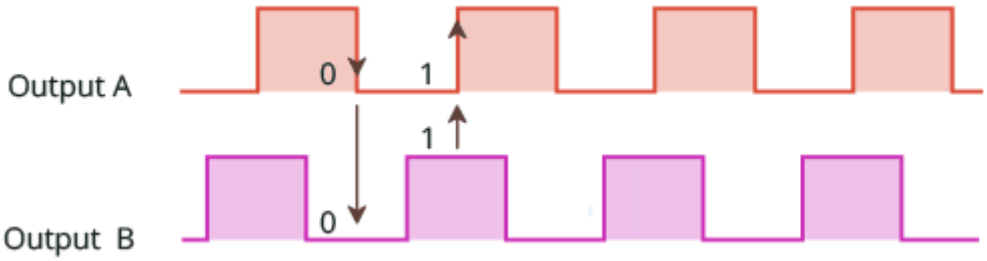
همان طور که در شکل ۴-۲۱ مشاهده می کنید با چرخش دیسک در جهت عقربه های ساعت ابتدا پایه A و سپس پایه B با بخش رسانی دیسک یعنی GND برخورد می کند و در صورتی که برعکس بچرخد یعنی خلاف عقربه های ساعت بچرخد ابتدا پایه B و سپس پایه A با زمین تماس پیدا خواهند کرد.

پایه های A و B به شیوه ای قرار داده شده اند که چرخش شفت روتاری انکودر با ترتیب خاصی به GND وصل شوند که دو سیگنال خروجی از A و B با یکدیگر اختلاف فاز ۹۰ درجه داشته باشند. به عنوان مثال اگر سیگنال پایه A تغییر کند و مقدار خروجی A با B برابر نباشد یعنی روتاری انکودر در جهت عقربه های ساعت چرخانده شده است.



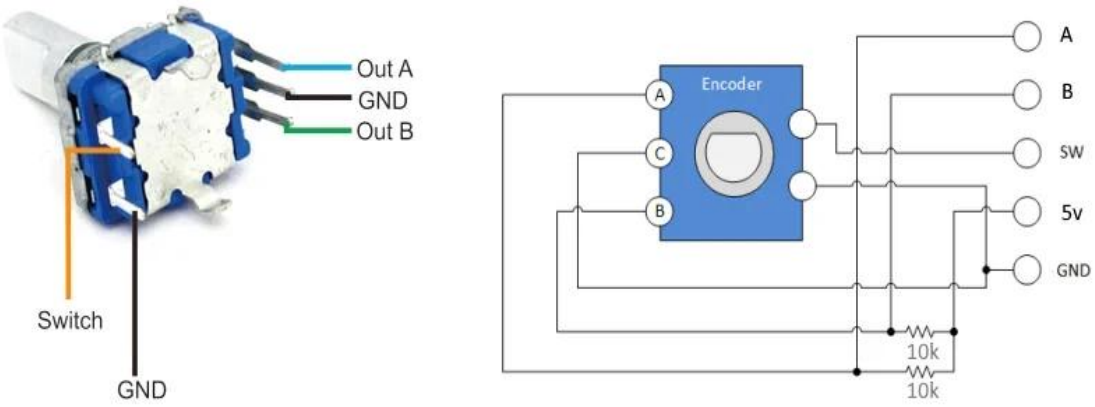
شکل ۴- ۲۲- خروجی پالس انکودر با چرخش در جهت عقربه های ساعت

و اگر سیگنال پایه A تغییر کند و مقدار خروجی A با B برابر باشد یعنی روتاری انکودر در جهت عکس عقربه های ساعت چرخانده شده است



شکل ۴- ۲۳- خروجی پالس انکودر با چرخش در جهت عکس عقربه های ساعت

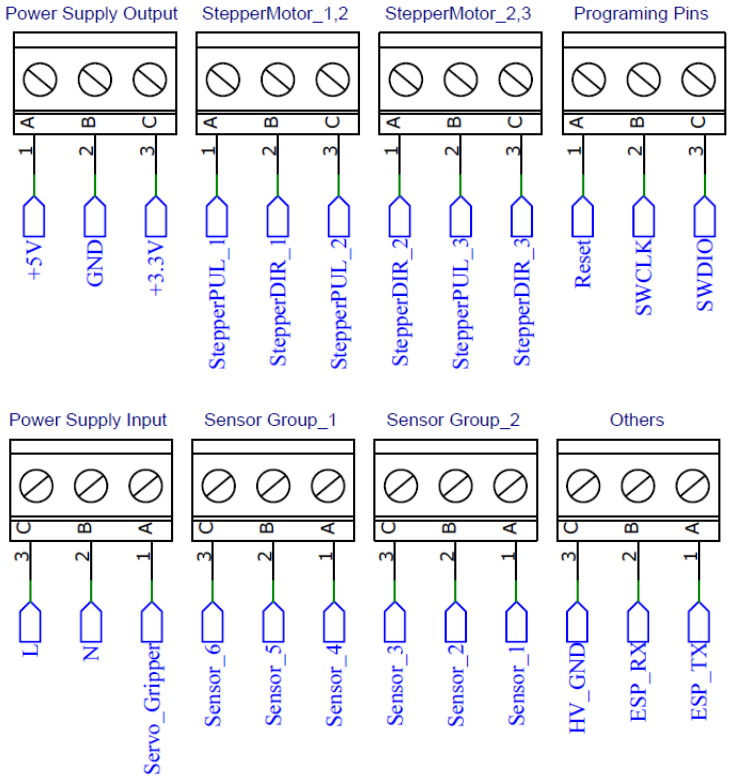
در صورتی که بخواهید از خود قطعه استفاده کنید ترتیب پایه ها مانند شکل ۴-۲۴ می باشد و برای اتصال به STM32 یا هر نوع میکروی دیگری باید طبق شماتیک موجود در تصویر زیر آن را پیاده کنید.



شکل ۴- ۲۴- شماتیک روتاری انکودر ولومی

۴-۳. بررسی شماتیک کنترلر ربات اسکارا

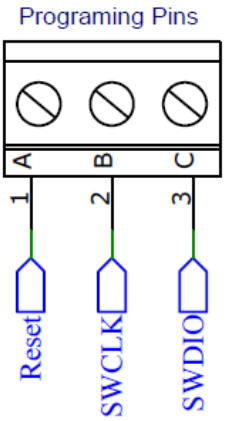
شماتیک کنترلر ربات اسکارا دارای ورودی و خروجی های متعددی می باشد که بخش اعظم آن مربوط به ترمینال خروجی تولید پالس و ترمینال وروردی سیگنال سنسورها است. در شکل ۴-۲۵ تمام ترمینال های ورودی و خروجی کنترلر مشخص شده است.



شکل ۴- ۲۵- ترمینال های ورودی و خروجی کنترلر

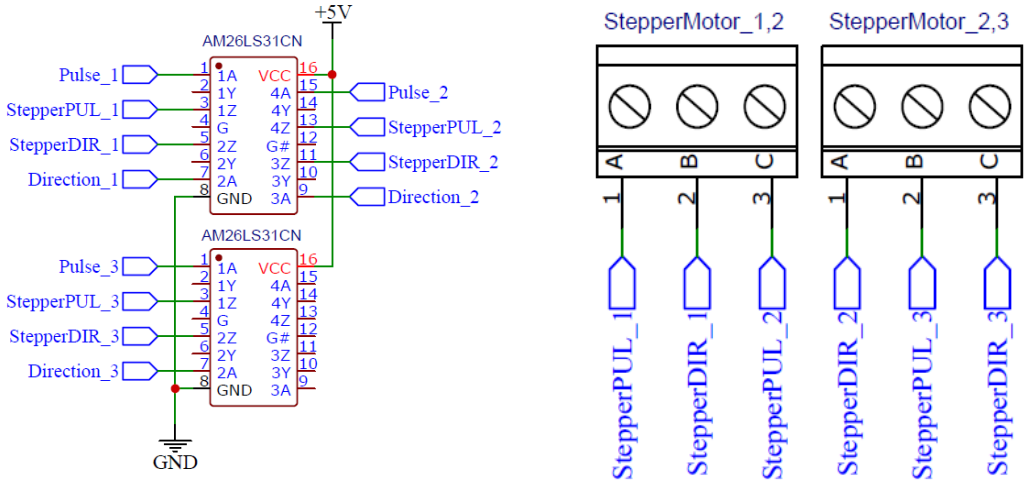
همانطور که در شکل ۴-۲۵ مشاهده می کنید ترمینال ها به دو بخش بالا و پایین کنترلر تقسیم شده است که هر بخش شامل ۱۲ ترمینال و در مجموع این کنترلر دارای ۲۴ ترمینال ورودی و خروجی می باشد.

در بخش بالایی کنترلر سه ترمینال برای پروگرام کردن میکروکنترلر SM32 تعبیه شده است که مستقیم به برد توسعه STM32F103C8T6 متصل شده اند.



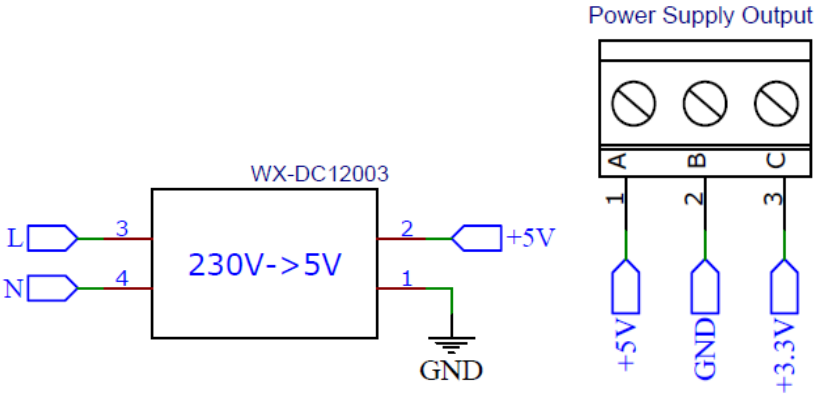
شکل ۴- ۲۶- ترمینال های پروگرام کردن میکرو STM32

در بخش بالایی کنترلر ۶ ترمینال برای کنترل استپر موتورهای سه محور ربات اسکارا تعبیه شده است که به واسطه آی سی لاین درایو AM26LS31 که قبلا آنرا معرفی کردیم به برد توسعه STM32F103C8T6 متصل شده اند.



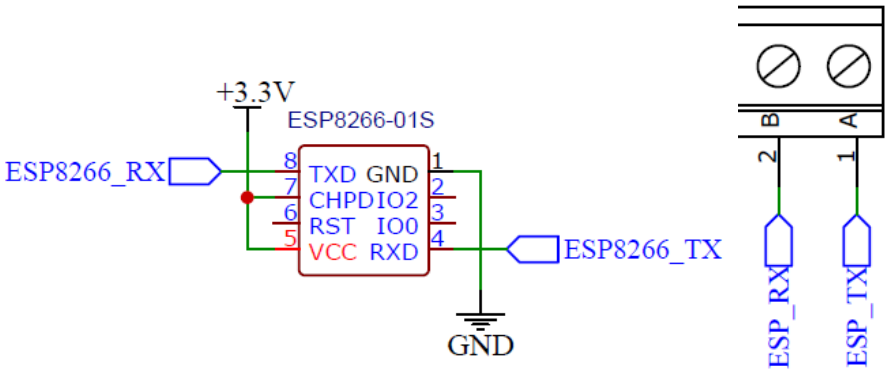
شکل ۴- ۲۷- ترمینال های کنترل استپر موتور و اتصالات آی سی لاین درایو

در بخش بالایی کنترلر سه ترمینال به عنوان منبع تغذیه خروجی برای مصارف خارج از کنترلر تعبیه شده است که خروجی ۳.۳ ولت آن از برد توسعه STM32F103C8T6 استفاده شده است و خروجی ۵ ولت آن از منبع تغذیه ورودی کنترلر استفاده می کند.



شکل ۴- ۲۸- ترمینال منبع تغذیه خروجی و ماژول منبع تغذیه ورودی کنترلر

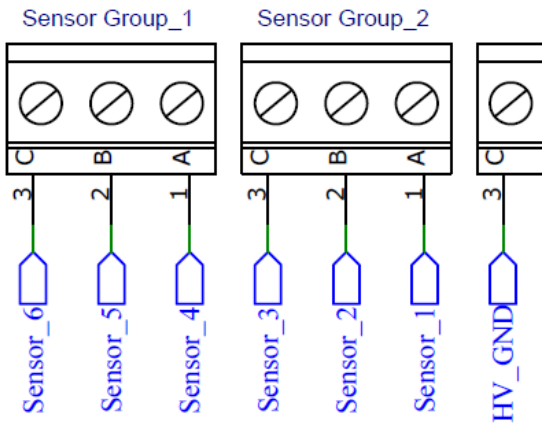
در بخش پایینی کنترلر دو ترمینال برای اتصال ماژول ESP8266 از طریق پورت سریال تعبیه شده است که البته این ماژول میتواند در داخل کنترلر تعبیه شود و از این ترمینال ها برای مصارف دیگر استفاده نمود اما در حال حاضر ماژول ESP8266 خارج از کنترلر استفاده شده است و به این دو ترمینال برای ارتباط با کنترلر نیاز مندیم.



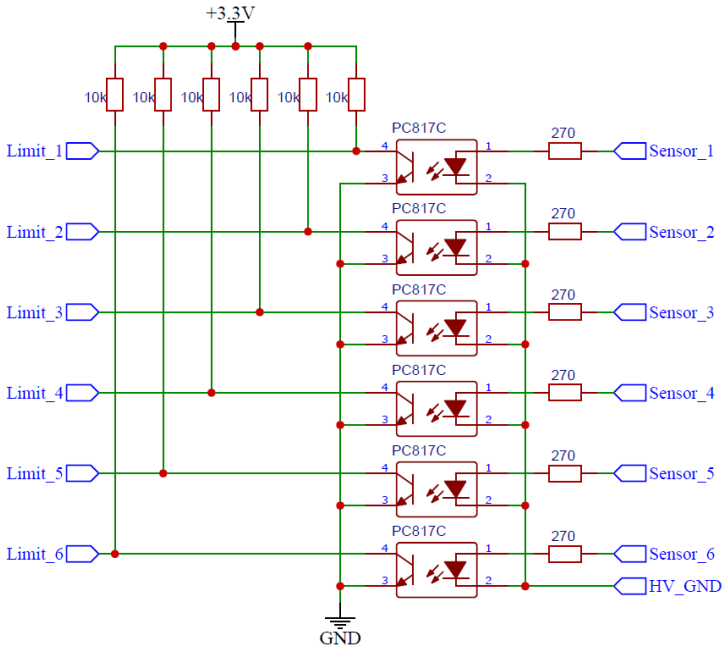
شکل ۴- ۲۹- ترمینال های ارتباط سریال و اتصالات ماژول ESP8266

در بخش پایینی کنترلر ۷ ترمینال برای دریافت سیگنال خروجی از سنسورهای واقع بر روی ربات اسکارا تعبیه شده است که به واسطه اپتوکوپلرهای PC817 به برد توسعه STM32F103C8T6 متصل شده اند. خروجی های این اپتوکوپلر با مقاومت ۱۰ کیلو اهم با تغذیه ۳.۳ ولت پول آپ شده اند و آرایش اتصال اپتوکوپلرها به میکروکنترلر بصورت NPN برقرار شده است.

به این نکته توجه کنید که ولتاژ ورودی قابل تحمل این اپتوکوپلرها حداکثر ۵.۵ ولت است و برای اتصال سنسورهای ربات اسکارا بدلیل اینکه سیگنال خروجی آنها ۲۴ ولت است باید حتما از یک مقاومت ۲.۲ کیلو اهم بصورت سری برای هر ورودی سنسور استفاده نمایید.

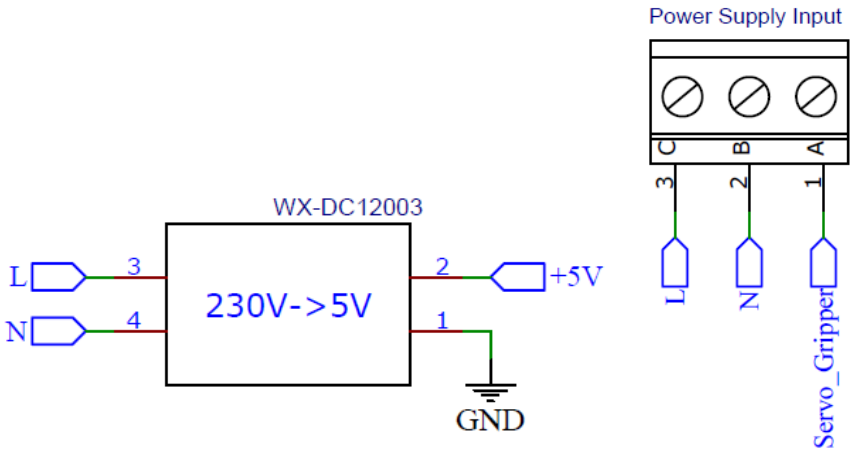


شکل ۴- ۳۰- ترمینال های ورودی سنسور



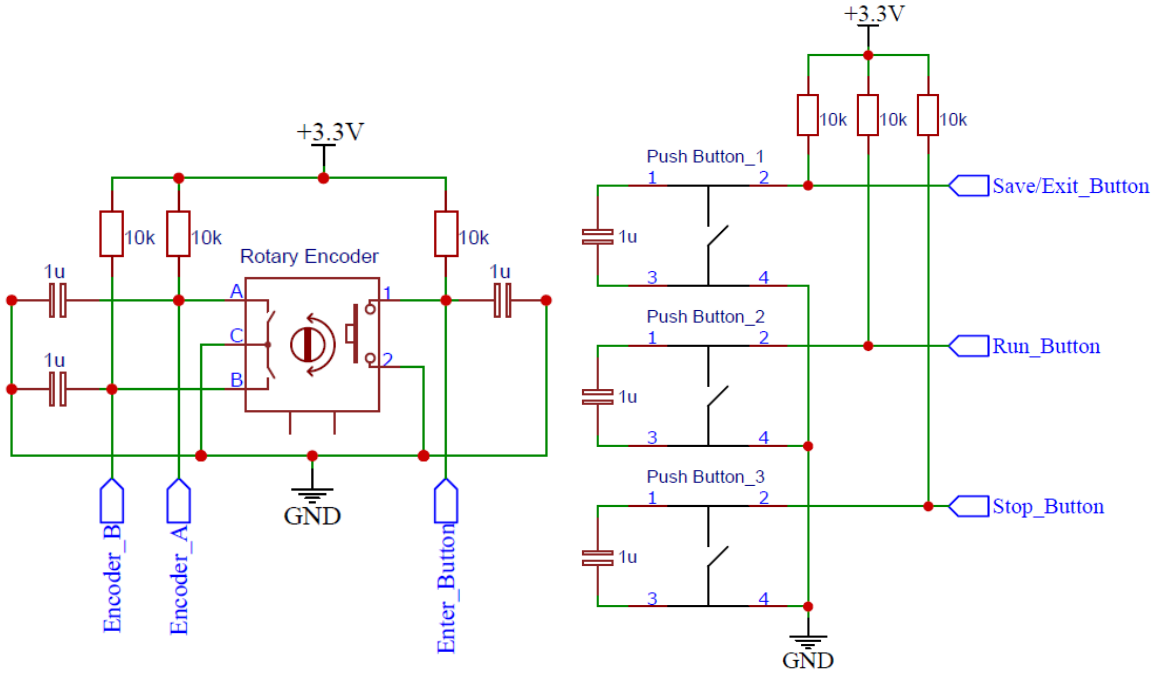
شکل ۴- ۳۱- شماتیک اتصالات اپتوکوپلر به برد توسعه STM32F103C8T6

در بخش پایینی کنترلر سه ترمینال یکی به عنوان کنترل سرو موتور برای اتصال گریپر و دو ورودی فاز و نول برای منبع تغذیه اصلی کنترلر تعبیه شده است که ترمینال کنترلر سروگریپر به صورت مستقیم به برد توسعه STM32F103C8T6 متصل شده است و دو ورودی فاز و نول به ماژول منبع تغذیه ۲۳۰ ولت به ۵ ولت برای تغذیه میکروکنترلر و دیگر مصارف متصل می شود.



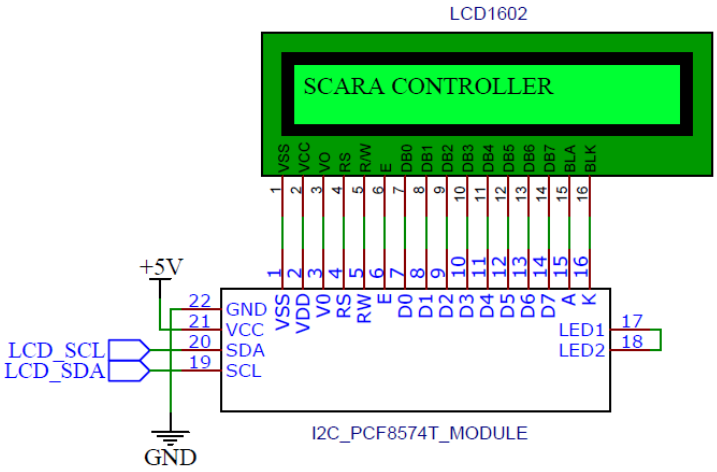
شکل ۴- ۳۲- ترمینال کنترل سرو موتور و ورودی منبع تغذیه کنترلر

در بخش داخلی کنترلر ربات اسکارا اتصالات دکمه ها و روتاری انکودر بصورت پول آپ به برد توسعه STM32F103C8T3 متصل شده است. همچنین از یک خازن ۱ میکروفاراد بین پایه ها برای کاهش نویز نیز تعبیه شده است.

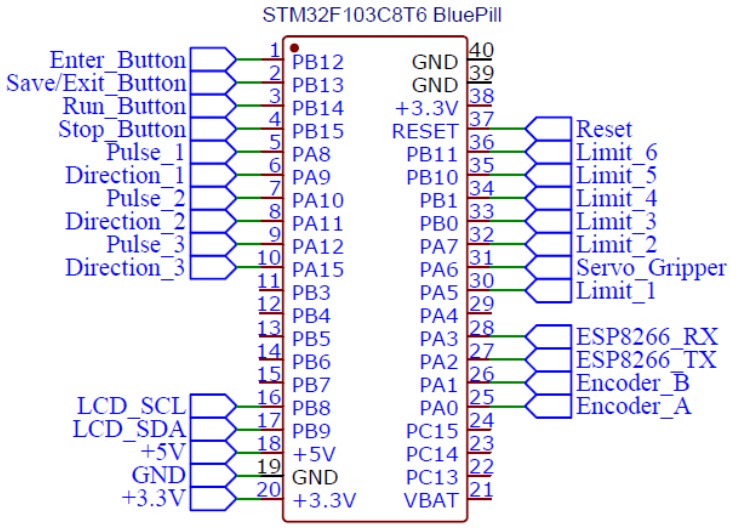


شکل ۴- ۳۳- شماتیک اتصالات کلیدها و روتاری انکودر به برد توسعه STM32F103C8T6

همانطور که قبلا توضیح دادیم LCD کاراکتری نیز به واسطه یک ماژول رابط I2C به میکروکنترلر متصل شده است. همچنین شماتیک برد توسعه STM32F103C8T6 در شکل ۴-۳۵ مشخص شده است.



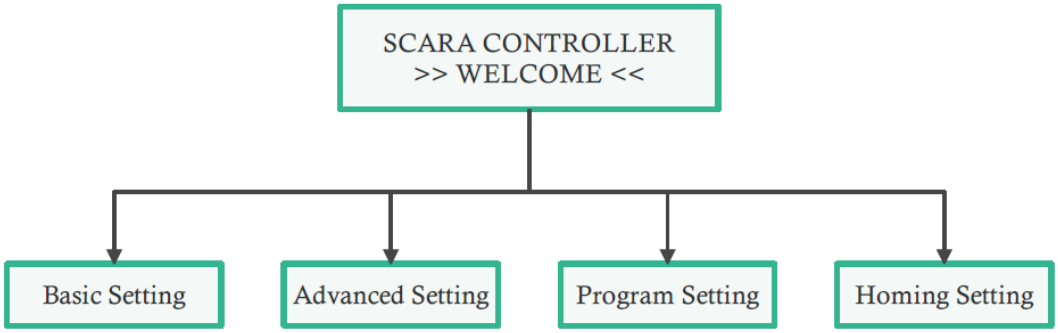
شکل ۴- ۳۴- شماتیک اتصال LCD کاراکتری همراه با رابط I2C



شکل ۴- ۳۵- شماتیک اتصالات برد توسعه STM32F103C8T6

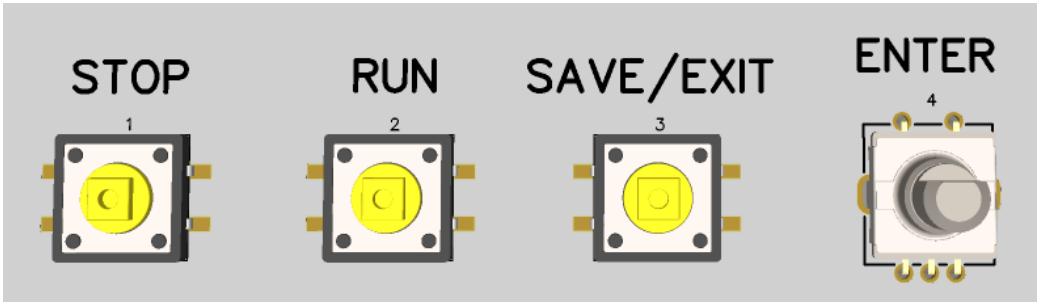
۴-۴. آموزش کار با رابط کاربری کنترلر

رابط کاربری کنترلر به چهار بخش اصلی تقسیم می شود که کاربر می تواند به راحتی عملکرد دستگاه را مشاهده و تنظیمات لازم را انجام دهد.



شکل ۴- ۳۶- منوی اصلی کنترلر ربات اسکارا

علاوه بر این، بر روی کنترلر سه دکمه و یک روتاری انکودر ولومی تعبیه شده است که برای هر یک وظیفه خاصی در نظر گرفته شده است و با استفاده از اینها می توان بین منو ها جابجا شد، وارد تنظیمات هر بخش شد و تغییرات لازم را برای هر یک اعمال نمود.



شکل ۴- ۳۷- دکمه های واقع بر روی کنترلر ربات اسکارا

همانطور که در شکل ۴-۳۷ مشاهده می نمایید از چپ به راست، دکمه STOP به عنوان یک شستی استپ اضطراری[1] در نظر گرفته شده است بدین معنا که با استفاده از این دکمه میتوان ربات را در هر لحظه از برنامه متوقف نمود اما توجه شود که در صورت استفاده از این دکمه، ربات موقعیت خود را از دست می دهد و نیاز هست که مجدد ربات عملیات Homing را انجام دهد.

دکمه RUN به عنوان یک شستی استارت در نظر گرفته شده است بدین معنا که با استفاده از این دکمه میتوان برنامه ای که برای ربات نوشته شده است را اجرا نمود و فقط در منوی Program Setting میتوان از آن استفاده نمود.

[1] Emergency Stop

دکمه SAVE / EXIT برای ذخیره اطلاعات تنظیم شده در هر بخش و خارج شدن از بخش مربوطه در نظر گرفته شده است و دکمه ENTER که در واقع همان دکمه فشاری روتاری انکودر ولومی می باشد، میتوان وارد هر بخش از منوی کنترلر شد. همچنین با چرخش روتاری انکودر نیز میتوان بین منوها یا زیر منوها نیز جابجا شد.

۴-۴-۱. معرفی منوی اصلی کنترلر[1]

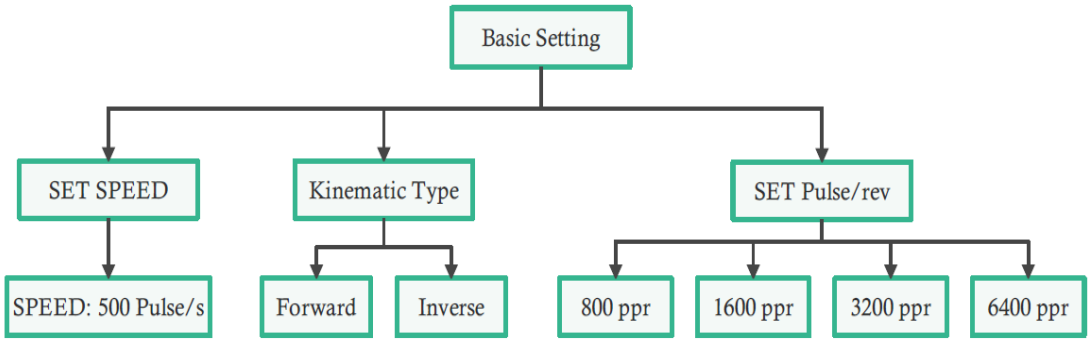
منوی اصلی کنترلر شامل چهار بخش تنظیمات می باشد که با چرخش روتاری انکودر میتوان بین این تنظیمات جابجا شد و با فشردن دکمه روتاری انکودر نیز میتوان وارد این تنظیمات شد.

شکل ۴- ۳۸- معرفی منوی اصلی کنترلر

۴-۴-۲. معرفی منوی تنظیمات پایه[2]

بخش اول منوی اصلی، تنظیمات پایه می باشد که در این بخش میتوان یکسری پارامترها را بصورت کلی و همگانی تنظیم نمود. در شکل ۴-۳۹ تمام زیر مجموعه های این منو را مشاهده می کنید. این منو شامل سه بخش تنظیم سرعت، انتخاب نوع سینماتیک ربات و تعیین تعداد پالس بر دور[3] بر اساس میزان تنظیمی روی درایو استپر موتور می باشد.

[1] Main Menu
[2] Basic Setting Menu
[3] Pulse pre-Revolution



شکل ۴- ۳۹- زیر مجموعه های منوی تنظیمات پایه

➢ بخش SET SPEED

در این بخش شما می توانید سرعت حرکت ربات را بر حسب پالس بر ثانیه تعیین کنید این میزان بیانگر فرکانس پالس تولیدی توسط کنترلر می باشد که به واسطه درایو به استپرموتور اعمال می شود.

شکل ۴- ۴۰- تنظیم سرعت حرکت ربات

در این بخش سرعت، توسط کاربر می تواند بطور معمول از مقدار حداقلی ۱۰ تا حداکثر ۹۹۹۹ پالس بر ثانیه تنظیم شود، اما باید توجه کنید که بدلیل محدودیت های مکانیکی ربات، حداقل سرعت قابل تنظیم در این بخش باید ۲۰۰ هرتز یا پالس بر ثانیه و حداکثر سرعت مجاز نیز باید در محدوده ۸۰۰ پالس بر ثانیه باشد، در غیر اینصورت موتور دچار لرزش شده و پالس گم می کند.

برای تنظیم سرعت باید از روتاری انکودر استفاده نمایید، بدین صورت که از چرخش آن برای افزایش و کاهش اعداد و از دکمه فشاری آن برای جابجا شدن بین ارقام که در بالا سمت راست با یک ضریب از ۱۰ تا ۱۰۰۰ طبق شکل ۴-۴۰ مشخص می شود، استفاده کرد.

دقت شود که برای خروج نیز طبق توضیحات پیشین باید از دکمه SAVE / EXIT استفاده نمایید، همچنین باید گفت که این مقدار تنظیمی برای همه استپر موتورهای ربات اسکارا تنظیم می شود.

➢ بخش Kinematic Type

در این بخش شما می توانید نوع سینماتیک ربات را انخاب نمایید. البته لازم به ذکر است که در فصل بعدی در مورد این موضوع بیشتر توضیح خواهیم داد اما برای شناخت اولیه باید بدانید که ما دو نوع سینماتیک داریم، یکی سینماتیک مستقیم و دیگری سینماتیک معکوس.

در سینماتیک مستقیم موقعیت های حرکتی ربات بر حسب زاویه مفصل ها بیان می شود یعنی با تنظیم زاویه هر مفصل، موقعیت نهایی[1] ربات، بدون دانستن موقعیت کارتزین یا XY انجام می شود. اما در سینماتیک معکوس موقعیت های حرکتی ربات بدون دانستن مقدار زاویه مفاصل و با دانستن موقعیت نهایی بر حسب XY بیان می شود یعنی با تعیین موقعیت نهایی ربات بصورت کارتزین میتوان میزان زوایه قرارگیری مفاصل ربات را پیش بینی کرد.

شکل ۴- ۴۱- تعیین نوع سینماتیک ربات

برای انتخاب نوع سینماتیک ربات نیز باید از چرخش روتاری انکودر برای جابجایی بین گذینه ها و دکمه SAVE / EXIT برای ذخیره و خروج از این بخش استفاده کرد.

➢ بخش SET Pulse/rev

در این بخش شما می توانید تعداد پالس بر دوری که بر روی درایو استپر موتور تنظیم کرده اید را در اینجا تعیین نمایید. همانطور که در فصل قبل توضیح دادیم استپر موتور متناسب با ساختاری که دارد دارای یک میزان گام می باشد که بر حسب زاویه بیان می شود یعنی با توجه به اینکه گام همه استپر موتورهای این ربات، ۱.۸ درجه است برای چرخش یک دورکامل ۳۶۰ درجه به میزان ۲۰۰ پالس احتیاج خواهیم داشت و از آنجاییکه این تعداد پالس، دقت بسیار پایینی را در اختیار استفاده کننده قرار می دهد نیاز هست که بواسطه درایو این میزان را با قابلیت میکرو استپینگ به واحد های ریزتر تبدیل نماییم، در این حالت هرچه تعداد پالس بر دور بیشتر باشد میزان دقت حرکت ربات نیز افزایش می یابد، در عوض احتمال از دست رفتن پالس های اعمالی به ربات نیز افزایش پیدا می کند.

[1] End Effector

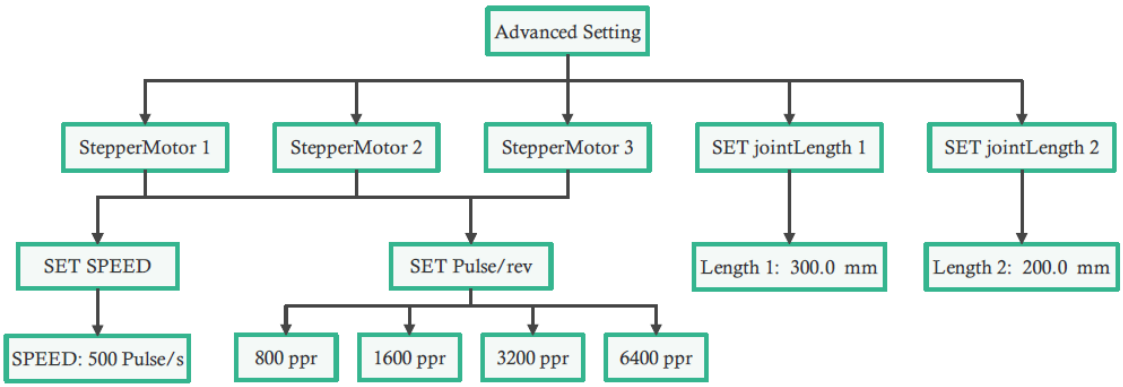
برای مثال فرض کنید که مقدار پالس بر دور را روی مقدار ۳۲۰۰ تنظیم نموده اید این بدین معناست که برای چرخش هر دور کامل استپر موتور این میزان پالس مورد نیاز است، حال با تقسیم ۳۶۰ بر این تعداد پالس، میزان زاویه چرخش استپر موتور به ازای هر پالس بدست می آید یعنی گام استپر موتور با این روش از ۱.۸ درجه به میزان ۰.۱۱۲۵ درجه به ازای هر پالس کاهش پیدا کرده است.

شکل ۴- ۴۲- تعیین تعداد پالس بر دور ربات

برای انتخاب تعداد پالس بر دور ربات نیز باید از چرخش روتاری انکودر برای جابجایی بین گذینه ها و دکمه SAVE / EXIT برای ذخیره و خروج از این بخش استفاده کرد. همچنین باید گفت که این مقدار تنظیمی برای همه استپر موتورهای ربات اسکارا تنظیم می شود.

۴-۴-۳. معرفی منوی تنظیمات پیشرفته[1]

بخش دوم منوی اصلی، تنظیمات پیشرفته می باشد که در این بخش میتوان پارامترهایی که در بخش تنظیمات پایه بصورت کلی تنظیم می شد را در این بخش بصورت جزئی تر تنظیم نمود. در شکل ۴-۴۳ تمام زیر مجموعه های این منو را مشاهده می کنید. این منو شامل پنج بخش تنظیم پارامترهای سرعت و پالس بر دور برای هر سه استپر موتور و تنظیم طول هر دو بازو یا مفصل ربات اسکارا می باشد.



شکل ۴- ۴۳- زیر مجموعه های منوی تنظیمات پیشرفته

[1] Advanced Setting Menu

بخش Stepper Motor 1 , 2 , 3

در این بخش شما می توانید همانند بخش تنظیمات پایه کلیه پارامترهای سرعت و پالس بر دور را بصورت اختصاصی و جداگانه برای هر یک از استپر موتورها تنظیم نمایید.

شکل ۴- ۴۴- تنظیم پارامترهای سرعت و پالس بر دور برای هر استپر موتور

بخش SET joint Length 1 , 2

در این بخش شما می توانید طول هر یک از لینک ها یا مفاصل ربات را تنظیم کنید. این مقدار در محاسبات سینماتیک معکوس استفاده خواهد شد به همین خاطر ما به طول دقیق هر بازو احتیاج خواهیم داشت.

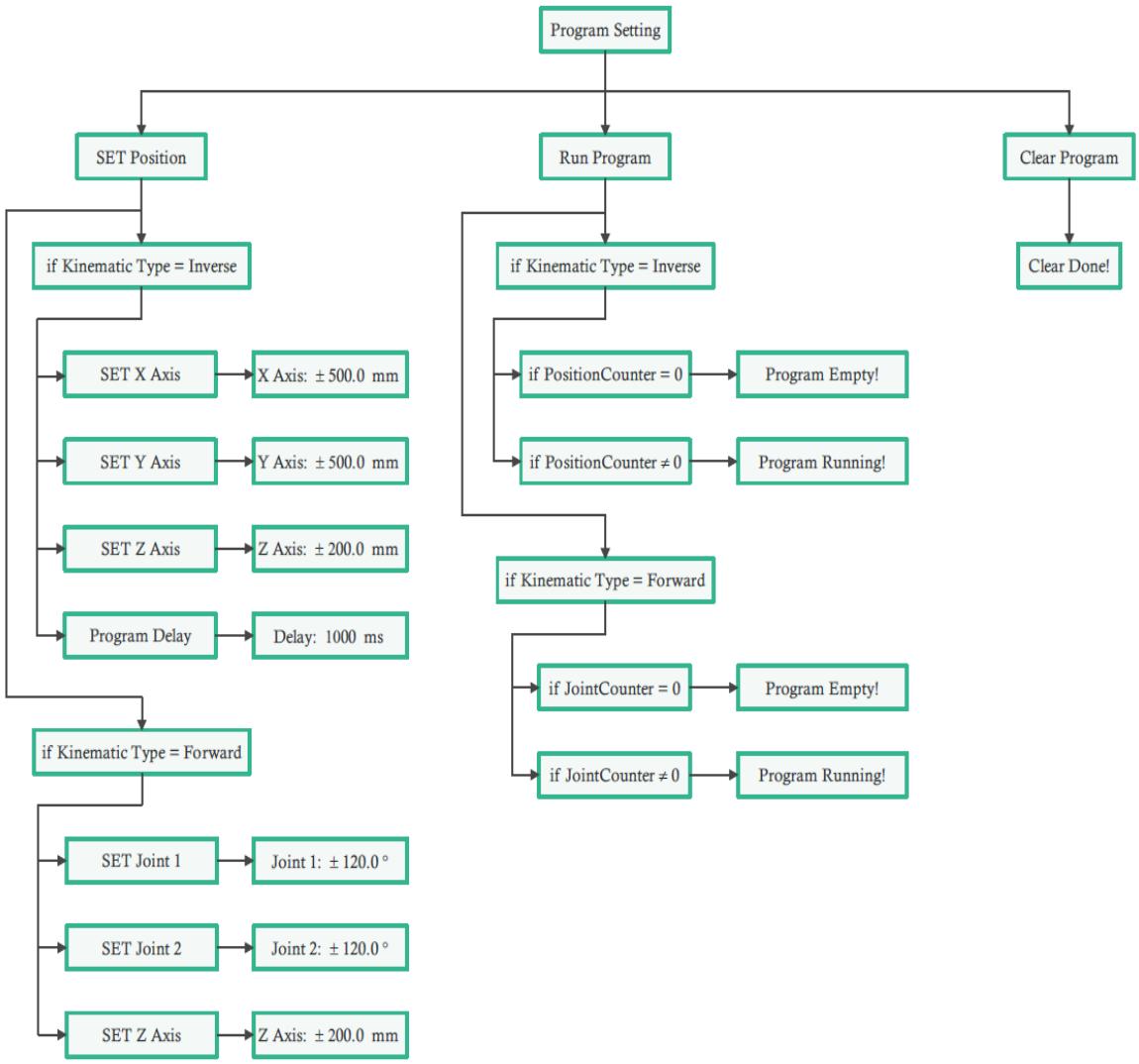
شکل ۴- ۴۵- تنظیم طول هر بازوی ربات اسکارا

برای تنظیم طول هر بازو همانند بخش های دیگر تنظیمات باید از روتاری انکودر استفاده نمایید، بدین صورت که از چرخش آن برای افزایش و کاهش اعداد و از دکمه فشاری آن برای جابجا شدن بین ارقام که در بالا سمت راست با یک ضریب از ۰.۱ تا ۱۰۰ طبق شکل ۴-۴۵ مشخص می شود، استفاده کرد. برای خروج نیز طبق توضیحات پیشین باید از دکمه SAVE / EXIT استفاده نمایید.

دقت کنید که این بخش پارامترها بصورت پیشفرض تعیین شده اند و نیازی به تنظیم هر بار آنها بعد از روشن شدن کنترلر و ربات نیست. بنابراین اگر شرایط عملکردی و ساختاری ربات تغییر پیدا کرد در صورت لزوم می توانید این پارامترها را بر حسب نیاز خود تغییر دهید.

۴-۴-۴. معرفی منوی تنظیمات برنامه[1]

بخش سوم منوی اصلی، تنظیمات برنامه می باشد که در این بخش میتوان موقعیت های ربات را متناسب با نوع سینماتیک ربات تنظیم و برنامه تنظیمی ربات را اجرا نمود. در شکل ۴-۴۶ تمام زیر مجموعه های این منو را مشاهده می کنید. این منو شامل سه بخش تنظیم موقعیت های ربات، اجرای برنامه تنظیمی ربات و پاک کردن برنامه یا کلیه موقعیت های تنظیم شده ربات می باشد.



شکل ۴- ۴۶- زیر مجموعه های منوی تنظیمات برنامه

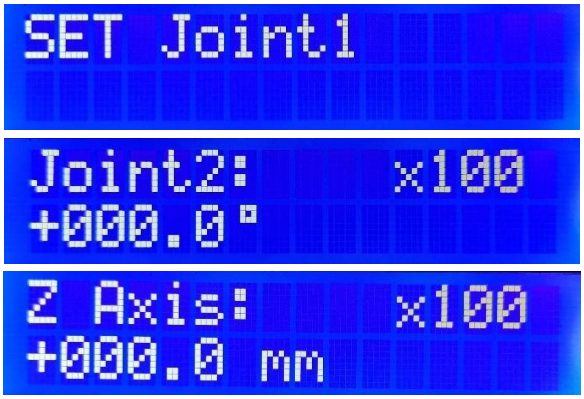
[1] Program Setting Menu

➢ بخش SET Position

در این بخش شما می توانید موقعیت های ربات را متناسب با نوع سینماتیک ربات که در بخش تنظیمات پایه انتخاب شد تنظیم نمایید. در ابتدا که وارد این بخش می شود با شماره موقعیت مربوطه مواجه خواهید شد که اگر سینماتیک مستقیم را انتخاب کرده باشید، این شمارش از صفر و اگر سینماتیک معکوس را انتخاب کرده باشید، این شمارش از یک شروع می شود. همچنین بصورت پیشفرض می توانید تا ۲۵ موقعیت را در این بخش ثبت نمایید.

شکل ۴- ۴۷- تنظیم موقعیت های ربات

پس از وارد شدن به هر یک از موقعیت ها، اگر نوع سینماتیک ربات را از نوع مستقیم انتخاب کرده باشید، می توانید زوایای مفاصل یک و دو و همچنین میزان جابجایی محور Z ربات را تنظیم نمایید.



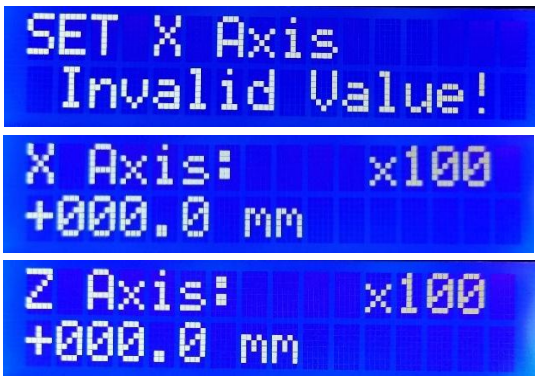
شکل ۴- ۴۸- تنظیم زوایای مفاصل یک و دو و میزان جابجایی محور Z ربات

در این بخش زوایا، توسط کاربر می تواند بطور معمول از مقدار حداقلی ۰.۱ تا حداکثر ۹۹۹ درجه تنظیم شود، اما باید توجه کنید که بدلیل محدودیت های مکانیکی ربات، حداقل زاویه قابل تنظیم در این بخش باید ۰.۱ درجه و حداکثر زاویه مجاز نیز باید در محدوده ای که در بخش هومینگ تعیین شده است باشد. در بخش تنظیمات هومینگ توضیحات بیشتری در این مورد خواهیم داد.

برای تنظیم زوایا همانند بخش های دیگر تنظیمات باید از روتاری انکودر استفاده نمایید، بدین صورت که از چرخش آن برای افزایش و کاهش اعداد و از دکمه فشاری آن برای جابجا شدن بین ارقام که در بالا سمت راست با یک ضریب از ۰.۱ تا ۱۰۰ طبق شکل ۴-۴۸ مشخص می شود، استفاده کرد و همچنین از دکمه RUN نیز می توانید برای تغییر علامت زاویه استفاده نمایید.

همانند تنظیم زوایا، میزان جابجایی محور Z نیز توسط کاربر می تواند بطور معمول از مقدار حداقلی ۰.۱ تا حداکثر ۹۹۹ میلی متر تنظیم شود، اما باید توجه کنید که بدلیل محدودیت های مکانیکی ربات، حداقل میزان جابجایی قابل تنظیم در این بخش باید ۰.۱ میلی متر و حداکثر میزان جابجایی مجاز نیز باید در محدوده ای که در بخش هومینگ تعیین شده است باشد.

اگر نوع سینماتیک ربات را از نوع معکوس انتخاب کرده باشید، می توانید مختصات کارتزین یا XYZ و همچنین میزان تاخیر بین هر برنامه یا موقعیت را تنظیم نمایید.



شکل ۴- ۴۹- تنظیم مختصات کارتزین یا XYZ

این بخش نیز همانند بخش قبل تنظیم می شود، اما باید توجه کنید که بدلیل محدودیت های مکانیکی ربات، باید حداقل یکی از مختصات X یا Y ، در محدوده ۲۰۰ تا ۵۰۰ میلی متر تنظیم شده باشد در غیر اینصورت با خطا مواجه خواهید شد و موقعیت شما ثبت نمی شود.

در بخش تنظیم تاخیر می توانید یک فاصله زمانی بر حسب میلی ثانیه بین هر برنامه یا موقعیت تنظیمی اختصاص دهید. همچنین برای تنظیم تاخیر باید از روتاری انکودر استفاده نمایید، بدین صورت که از چرخش آن برای افزایش و کاهش اعداد و از دکمه فشاری آن برای جابجا شدن بین ارقام که در بالا سمت راست با یک ضریب از ۱۰ تا ۱۰۰۰ طبق شکل ۴-۵۰ مشخص می شود، استفاده کرد.

شکل ۴- ۵۰- تنظیم تاخیر بین برنامه ها یا موقعیت ها

➢ بخش Run Program

در این بخش شما می توانید برنامه تنظیم شده در مرحله قبل را بر روی ربات اجرا نمایید و با دکمه RUN واقع بر روی کنترلر این کار را انجام دهید. همانطور که قبلا توضیح دادیم دکمه STOP نیز به عنوان یک شستی استپ اضطراری[1] در نظر گرفته شده است بدین معنا که با استفاده از این دکمه میتوان ربات را در هر لحظه از برنامه متوقف نمود اما توجه شود که در صورت استفاده از این دکمه، ربات موقعیت خود را از دست می دهد و نیاز هست که مجدد ربات عملیات Homing را انجام دهد.

شکل ۴- ۵۱- اجرای برنامه تنظیمی بر روی ربات

دقت کنید که اگر هیچ برنامه ای در این بخش تنظیم نشده باشد، خطای شکل ۴-۵۲ بر روی صفحه LCD نمایش داده می شود.

شکل ۴- ۵۲- خطای عدم وجود برنامه برای اجرا

زمانی که برنامه ای در این بخش نیز تنظیم شده باشد، هشدار شکل ۴-۵۳ بر روی صفحه LCD نمایش داده می شود. همچنین توجه شود در هنگام اجرای برنامه هیچ تغییراتی را نمی توانید در کنترلر به غیر از متوقف کردن برنامه توسط دکمه STOP اعمال نمایید.

شکل ۴- ۵۳- هشدار اجرای برنامه

[1] Emergency Stop

- **بخش Clear Program**

در این بخش شما می توانید تمام برنامه یا موقعیت هایی که برای ربات تنظیم نموده اید را پاک و از ابتدا موقعیت یا برنامه جدیدی را تنظیم نمایید.

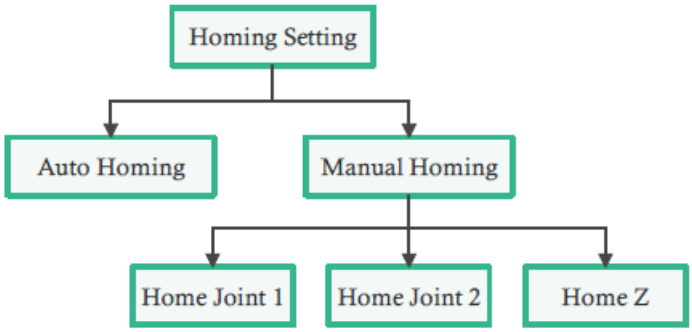
شکل ۴- ۵۴- پاک کردن برنامه تنظیمی ربات

برای پاک کردن برنامه یا موقعیت ها، فقط کافیست از دکمه فشاری روتاری انکودر استفاده نمود و پس از فشردن دکمه مربوطه، هشدار شکل ۴-۵۴ بر روی صفحه LCD نمایش داده می شود.

شکل ۴- ۵۵- هشدار پاک شدن موقعیت یا برنامه ها

۴-۴-۵. معرفی منوی تنظیمات هومینگ[1]

بخش چهارم منوی اصلی، تنظیمات هومینگ می باشد که در این بخش میتوان موقعیت صفر و محدوده مجاز حرکت محورهای ربات را تعیین نمود. در شکل ۴-۵۶ تمام زیر مجموعه های این منو را مشاهده می کنید. این منو شامل دو بخش هومینگ خودکار و هومینگ دستی می باشد.



شکل ۴- ۵۶- زیر مجموعه های منوی تنظیمات هومینگ

[1] Homing Setting Menu

بخش Auto Homing

در این بخش شما می توانید تنها با استفاده از دکمه فشاری روتاری انکودر عملیات هومینگ را بصورت خودکار انجام دهید. این عملیات بدین صورت انجام می شود که بازوهای ربات به ترتیب از بازوی اول در هر موقعیتی که قرار داشته باشند، ابتدا در جهت ساعتگرد حرکت می کنند تا زمانی که توسط سنسور اول تشخیص داده شوند، این بیانگر آن است که بازو یا مفصل ربات نمی تواند از زاویه مشخص شده بیشتر بچرخد. پس از برخورد بازوی ربات با سنسور اول، جهت آن معکوس شده و در جهت پاد ساعتگرد شروع به حرکت می کند تا زمانی که توسط سنسور بعدی تشخیص داده شود، در این حالت نیز مانند حالت قبل یک محدودیت زاویه ای برای بازوی ربات تعیین می شود و در نهایت بازوی ربات در موقعیت صفر یا هم راستا با محور X قرار می گیرد.

شکل ۴- ۵۷- انجام عملیات هومینگ بصورت خودکار

بخش Manual Homing

زمانی که در این بخش وارد شوید شما می توانید عملیات هومینگ برای تمام محورهای ربات اسکارا که در حالت خودکار انجام می شد، بصورت دستی انجام دهید.

شکل ۴- ۵۸- انجام عملیات هومینگ بصورت دستی

در این بخش نیز تنها با استفاده از دکمه فشاری روتاری انکودر می توانید عملیات هومینگ را بصورت دستی برای هر یک از محورهای ربات اسکارا انجام دهید.

۴-۵. معرفی انواع صفحات وب و پروتکل های ارتباطی

استفاده از صفحه وب به کاربران این امکان را می‌دهد که با هر دستگاهی که دارای مرورگر است (مانند گوشی هوشمند، تبلت یا کامپیوتر) به سادگی ربات را کنترل کرده یا وضعیت آن را مشاهده کنند. ترکیب قابلیت‌های صفحه وب و ماژول ESP8266، نه تنها ارتباط بلادرنگ و پایدار را فراهم می‌کند، بلکه توسعه و شخصی‌سازی پروژه‌ها را نیز آسان می‌سازد.

شکل ۴- ۵۹- معرفی انواع صفحات وب

ماژول ESP8266، به دلیل توانایی اتصال به شبکه‌های Wi-Fi، پشتیبانی از پروتکل‌های استاندارد وب و قیمت مناسب به ابزاری ایده‌آل برای ایجاد ارتباط بین ربات و وب سرور تبدیل شده است. این ماژول می‌تواند به عنوان یک سرور عمل کند و دستورات را از طریق مرورگر دریافت کرده و به کنترلر منتقل کند یا داده‌های کنترلی ربات را در صفحه وب نمایش دهد.

۴-۵-۱. صفحات وب آنلاین[1]

صفحات وب آنلاین به صفحاتی گفته می‌شود که برای عملکرد به اتصال دائمی به اینترنت و ارتباط با یک سرور خارجی نیاز دارند. این صفحات به کاربران اجازه می‌دهند از طریق یک مرورگر وب، دستگاه‌هایی مانند ربات‌ها یا سخت‌افزارهایی که از ماژول‌هایی مانند ESP8266 استفاده می‌کنند را کنترل کنند. در این روش، تمام تبادل داده‌ها بین صفحه وب و ماژول ESP8266 از طریق سرور انجام می‌شود، که به کاربران امکان کنترل ربات یا دستگاه خود از هر نقطه جهان را می‌دهد. این صفحات به ویژه برای پروژه‌هایی که نیاز به کنترل از راه دور یا اشتراک‌گذاری داده‌ها با چندین کاربر را دارند، ایده‌آل هستند.

[1] Online Web Pages

۴-۵-۱-۱. نحوه ارتباط با ماژول ESP8266

۱) ابتدا ماژول ESP8266 به یک شبکه Wi-Fi متصل می‌شود، سپس از پروتکل‌هایی مانند HTTP ، WebSocket و یا MQTT برای برقراری ارتباط با سرور استفاده می‌کند.

۲) دستورات کاربر از طریق صفحه وب به سرور ارسال می‌شوند.

۳) سرور این دستورات را به ماژول ESP8266 منتقل می‌کند.

۴) پاسخ‌های ماژول ESP8266 به سرور ارسال شده و سپس به صفحه وب بازمی‌گردد.

۴-۵-۱-۲. ویژگی های صفحات وب آنلاین

- **کنترل از راه دور**

امکان کنترل دستگاه یا ربات از هر نقطه جهان با اتصال اینترنت.

- **دسترسی چند کاربره**

چندین کاربر می‌توانند به صورت هم زمان با صفحه وب متصل شوند.

- **ادغام با سرویس‌های ابری**

امکان ذخیره‌سازی داده‌ها و ارائه گزارشات در سرورهای ابری مانند Firebase یا Google Cloud

- **ارتباط بلادرنگ**

با استفاده از WebSocket یا MQTT ، می‌توان داده‌ها را بدون تأخیر به‌روزرسانی کرد.

۴-۵-۱-۳. چالش های صفحات وب آنلاین

- **نیاز به اتصال پایدار اینترنت**

در صورت قطع اینترنت، کنترل ربات امکان‌پذیر نیست.

- **مشکلات امنیتی**

به دلیل دسترسی از راه دور و قابلیت ارتباط از هر نقطه، صفحات وب آنلاین باید از اقدامات امنیتی پیشرفته مانند رمزنگاری داده‌ها و احراز هویت قوی برخوردار باشند، در غیر این صورت ممکن است در معرض حملاتی مانند هک یا نفوذ قرار بگیرند.

- **وابستگی به سرور**

اگر سرور از دسترس خارج شود، ارتباط با ماژول ESP8266 قطع می‌شود.

- **پیچیدگی بیشتر**

پیاده‌سازی نیازمند مهارت در مدیریت سرور و پروتکل‌های ارتباطی است.

۴-۵-۲. صفحات وب آفلاین[1]

این نوع صفحات برای زمانی مناسب هستند که نیاز به اتصال به اینترنت نباشد و کاربر فقط در محدوده شبکه Wi-Fi محلی که توسط ماژول ESP8266 ارائه می‌شود، کنترل ربات را انجام دهد. این صفحات تمام عملکرد خود را به صورت آفلاین و بدون نیاز به اتصال به سرور خارجی انجام می‌دهند.

۴-۵-۲-۱. نحوه ارتباط با ماژول ESP8266

۱) کاربر به شبکه Wi-Fi که توسط ESP8266 ایجاد شده، متصل می‌شود.
۲) یک صفحه وب شامل HTML ، CSS و JavaScript در خود ماژول ESP8266 میزبانی می‌شود.
۳) دستورات کاربر به طور مستقیم از طریق صفحه وب به ماژول ESP8266 ارسال می‌شوند و نیازی به سرور خارجی نیست.

۴-۵-۲-۲. ویژگی های صفحات وب آفلاین

- **بدون نیاز به اینترنت**

این سیستم در محیط‌هایی که اینترنت در دسترس نیست، کاربرد دارد.

- **ساده‌تر از صفحات آنلاین**

به دلیل عدم نیاز به سرور خارجی، پیاده‌سازی ساده‌تر است.

- **عملکرد سریع‌تر**

بدلیل اینکه ارتباطات به طور مستقیم بین صفحه وب و ESP8266 انجام می‌شود و نیازی به ارسال درخواست به سرور نیست عملکرد، بسیار سریع‌تر از صفحات آنلاین خواهد بود.

۴-۵-۲-۳. چالش های صفحات وب آفلاین

- **محدودیت در دسترسی**

از آنجایی که ماژول ESP8266 به عنوان نقطه دسترسی[2] عمل می‌کند، کاربر باید در محدوده شبکه Wi-Fi که توسط ماژول ESP8266 ایجاد شده قرار داشته باشد.

- **ارتباط تک کاربره**

در این حالت، فقط یک دستگاه می‌تواند به طور همزمان به ماژول ESP8266 متصل شود. این موضوع می تواند محدودیت‌هایی را در کاربردهای چند کاربره ایجاد می‌کند.

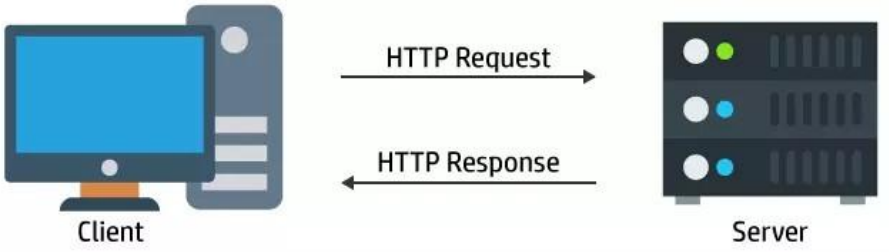
[1] Offline Web Pages
[2] Access Point

۴-۵-۳. روش های ارتباطی ماژول ESP8266 با صفحه وب

ماژول‌هایESP مانند ESP8266 و ESP32 قابلیت‌های متعددی برای ارتباط با صفحات وب دارند. این ارتباطات معمولاً از طریق پروتکل‌های مختلف شبکه انجام می‌شوند. در ادامه، انواع روش‌های ارتباط این ماژول‌ها با صفحات وب توضیح داده شده است.

۴-۵-۳-۱. ارتباط HTTP (سرور و کلاینت)

HTTP یک پروتکل استاندارد و رایج برای تبادل اطلاعات در وب است که بر اساس مدل درخواست و پاسخ [1] عمل می‌کند. ماژول ESP8266 می‌تواند در دو نقش مختلف (کلاینت یا سرور) در این پروتکل فعالیت کند.



شکل ۴- ۶۰- ارتباط HTTP

- **حالت کلاینت**

ماژول ESP8266 به عنوان یک کلاینت HTTP عمل می‌کند و درخواست‌هایی مانند GET یا POST را به یک سرور ارسال می‌کند.

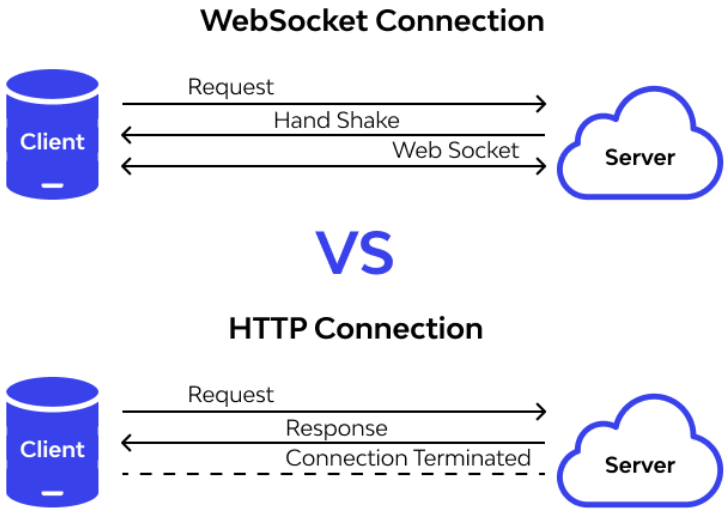
- **حالت سرور**

در این حالت، ماژول ESP8266 به عنوان یک سرور HTTP عمل می‌کند و صفحات وب یا داده‌ها را در پاسخ به درخواست‌های کاربران (مرورگرها) ارائه می‌دهد.

۴-۵-۳-۲. ارتباط WebSocket

WebSocketیک پروتکل ارتباطی است که امکان برقراری ارتباط مداوم و دوطرفه بین سرور و کلاینت را فراهم می‌کند. برخلاف HTTP که مبتنی بر ارسال درخواست و دریافت پاسخ است، WebSocket یک ارتباط دائمی ایجاد می‌کند که در آن هر دو طرف (ماژول ESP8266 و مرورگر یا سرور) می‌توانند به صورت مستقل داده ارسال یا دریافت کنند.

[1] Request / Response

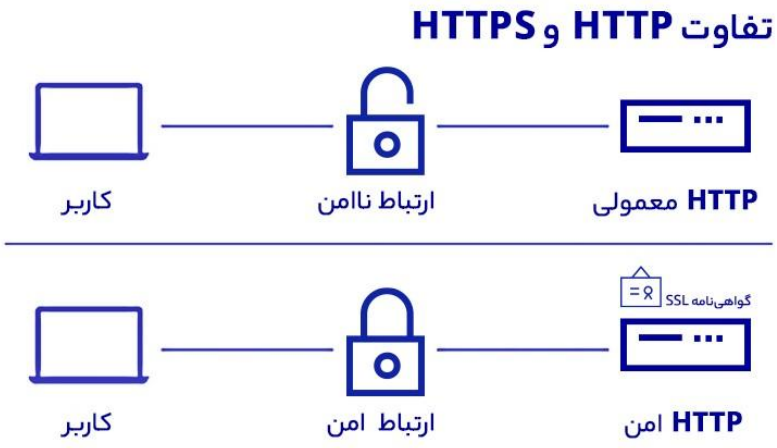


شکل ۴- ۶۱- ارتباط HTTP در مقابل ارتباط WebSocket

این روش برای مواردی که به انتقال بلادرنگ داده نیاز است، بسیار مناسب است. ارتباط WebSocket معمولاً در پروژه‌هایی بکار می‌رود که نیاز به بروزرسانی لحظه‌ای اطلاعات یا کنترل سریع دستگاه‌ها وجود دارد.

۴-۵-۳-۳. ارتباط HTTPS

HTTPS نسخه امن از پروتکل HTTP است که از رمزنگاری SSL برای ایمن سازی داده‌های در حال انتقال استفاده می‌کند. این پروتکل به ویژه برای مواقعی که اطلاعات حساس (مانند رمزهای عبور یا داده‌های خصوصی) باید منتقل شوند، بسیار حیاتی است.

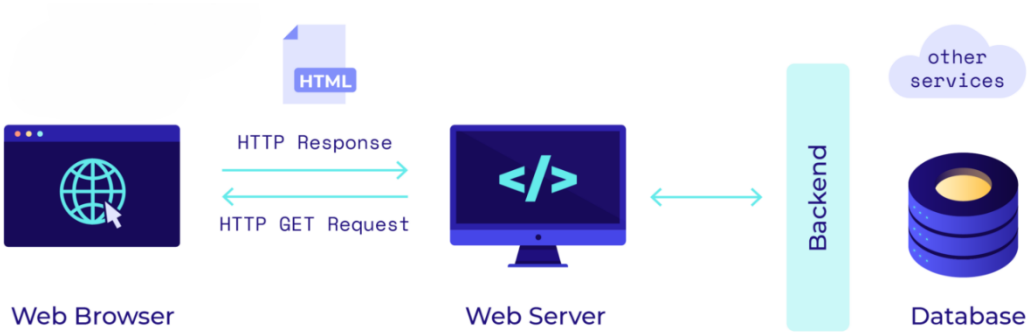


شکل ۴- ۶۲- ارتباط HTTP در مقابل ارتباط HTTPS

در ارتباط HTTPS ، داده‌ها قبل از ارسال، رمزگذاری شده و تنها توسط گیرنده قابل رمزگشایی هستند. ماژول ESP8266 می‌تواند به عنوان کلاینت HTTPS عمل کرده و به سرورهای امن متصل شود یا داده‌های امن را از آن‌ها دریافت کند. پیاده‌سازی HTTPS نیازمند تنظیم گواهینامه‌های SSL و استفاده از کتابخانه‌های مناسب است که حجم و توان پردازشی ماژول ESP8266 را نیز در نظر می‌گیرند.

۴-۶. معرفی و بررسی صفحه وب کنترلر

در این پروژه، برای کنترل ربات از یک صفحه وب آفلاین استفاده شده است و ارتباط بین ماژول ESP8266 و صفحه وب از طریق پروتکل HTTP برقرار می‌شود. در این ساختار، ماژول ESP8266 به‌عنوان سرور HTTP عمل می‌کند و صفحه وب نیز به عنوان کلاینت به آن متصل می‌شود.



شکل ۴- ۶۳- ارتباط صفحه وب با ماژول ESP8266

زمانی که صفحه وب فعال شده و درخواست‌هایی به ماژول ارسال می‌کند، ماژول ESP8266 داده‌ها را پردازش کرده و اطلاعات مورد نظر را به صفحه وب ارسال می‌کند یا دستورات کلاینت را اجرا می‌کند. ویژگی های این طراحی عبارت اند از:

- **کاهش پردازش در سمت ماژول ESP8266**

به دلیل اینکه ماژول ESP8266 فقط زمانی داده‌ها را ارسال یا پردازش می‌کند که صفحه وب درخواست مشخصی را ارسال کرده باشد، بار پردازشی ماژول کاهش می‌یابد.

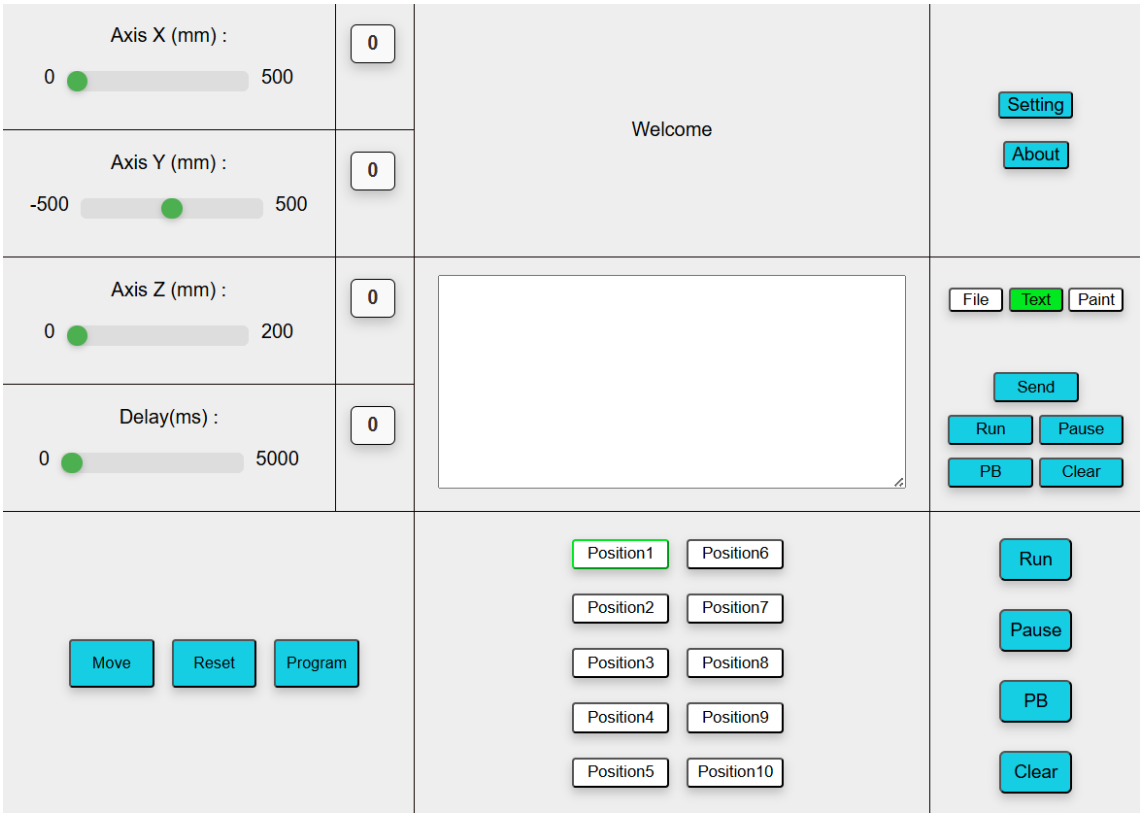
- **عملکرد مستقل کنترلر**

ماژول ESP8266 و کنترلر حتی بدون فعال‌سازی صفحه وب نیز قادر به انجام وظایف اصلی خود هستند و عملکرد سیستم وابسته به صفحه وب نیست.

- **سادگی پیاده‌سازی**

استفاده از پروتکل HTTP به عنوان یک روش استاندارد، فرآیند ارتباط و مدیریت داده‌ها را ساده و کارآمد می‌سازد.

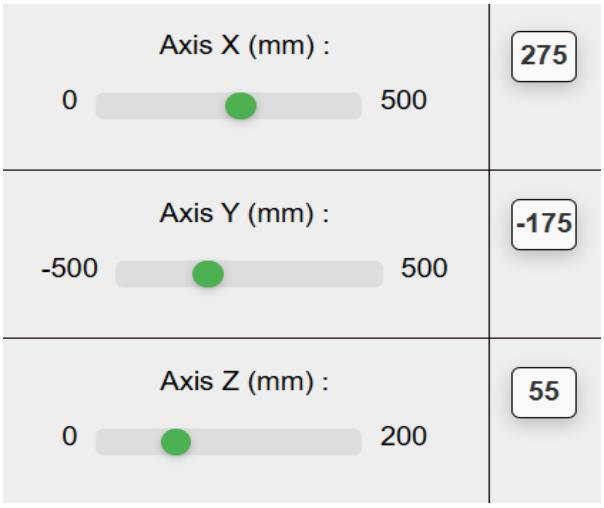
در ادامه به معرفی بخش های مختلف این صفحه وب خواهیم پرداخت.



شکل ۴- ۶۴- معرفی بخش های مختلف صفحه وب

۴-۶-۱. بخش مقداردهی مختصات

در این بخش چندین اسلایدر وجود دارد که شما می توانید مختصات حرکت ربات را در سه محور XYZ تنظیم نمایید. مختصاتی که در این بخش تنظیم می شود را می توان به چند حالت برای کنترلر ارسال کرد که در بخش های بعدی توضیح بیشتر داده می شود.



شکل ۴- ۶۵- بخش مقداردهی مختصات

۴-۶-۲. بخش دکمه های عملکردی

در این بخش چند دکمه وجود دارد که عملکرد آنها در ادامه توضیح داده شده است.

شکل ۴- ۶۶- بخش دکمه های عملکردی

- **عملکرد دکمه Move**

با استفاده از این دکمه، مختصاتی که در بخش قبل تنظیم شده‌اند، مستقیماً به کنترلر ربات ارسال می‌شوند و کنترلر، ربات را به مختصات تنظیم‌شده حرکت می‌دهد.

شکل ۴- ۶۷- عملکرد دکمه Move

- **عملکرد دکمه Reset**

با استفاده از این دکمه، مختصاتی که در بخش قبل تنظیم شده‌اند، به حالت پیش فرض باز می‌گردند یا مقادیر آنها به صفر تغییر می‌کنند.

شکل ۴- ۶۸- عملکرد دکمه Reset

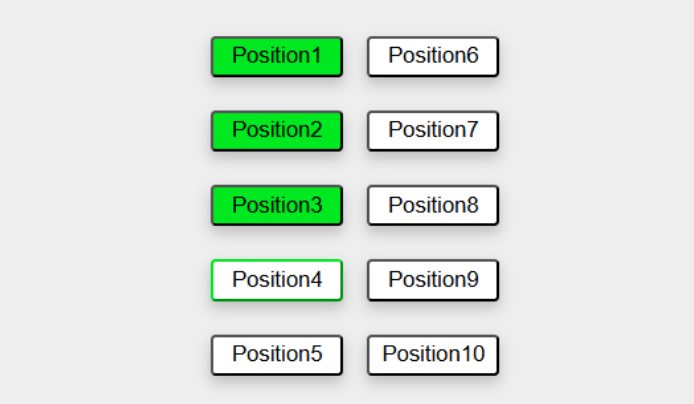
- **عملکرد دکمه Program**

با استفاده از این دکمه، مختصاتی که در بخش قبل تنظیم شده‌اند، در یکی از موقعیت ها ذخیره می‌شوند. سپس می‌توانید ربات را متناسب با موقعیت های ذخیره‌شده، به حرکت درآورید.

شکل ۴- ۶۹- عملکرد دکمه Program

۴-۶-۳. بخش نمایش موقعیت های ذخیره شده

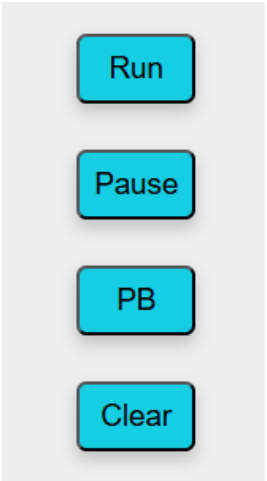
موقعیت‌هایی که در بخش مقداردهی مختصات ذخیره می‌کنید، در یکی از خانه‌های این بخش نمایش داده می شود. هر خانه‌ای که مختصات در آن ذخیره شود، رنگ آن تغییر می‌کند و با کلیک روی خانه‌های سبز (موقعیت‌های ذخیره‌شده) می‌توانید مقادیر تنظیم شده برای آنها را در بخش مقداردهی مختصات نیز مشاهده کنید.



شکل ۴- ۷۰- نمایش موقعیت های ذخیره شده

۴-۶-۴. دکمه های کنترلی

در این بخش با استفاده از دکمه های موجود، می‌توانید موقعیت‌های ذخیره‌شده را به ترتیب اجرا کنید، متوقف کنید، ادامه حرکت را از سر بگیرید یا حتی تمامی موقعیت‌های ذخیره شده را پاک کنید.



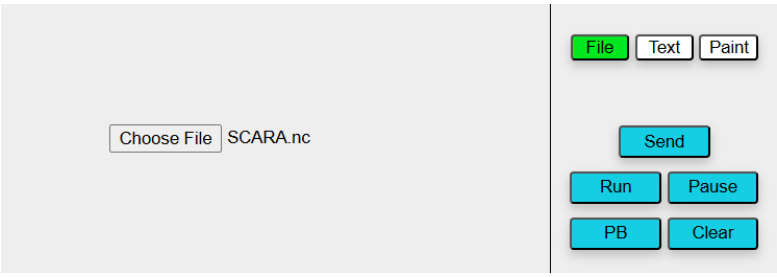
شکل ۴- ۷۱- دکمه های کنترلی

۴-۶-۵. بخش مدیریت کدها

در این بخش، می‌توانید نوع یا نحوه نوشتن کدها را تعیین کنید. این کار از طریق روش‌های زیر امکان‌پذیر است:

- **بارگذاری فایل Gcode**

در این حالت شما می توانید فایل Gcode خود را از منابع دیگر وارد کنید و از داده های آن، برای کنترل ربات استفاده نمایید.



شکل ۴- ۷۲- محل بارگذاری فایل Gcode

- **نوشتن دستی کدها**

در این حالت شما می توانید با دانستن دستورات Gcode آنها را بصورت دستی وارد نمایید و با ارسال آنها به کنترلر، فرمان مورد نظرتان را بر روی ربات اجرا کنید.

شکل ۴- ۷۳- محل نوشتن دستی کدها

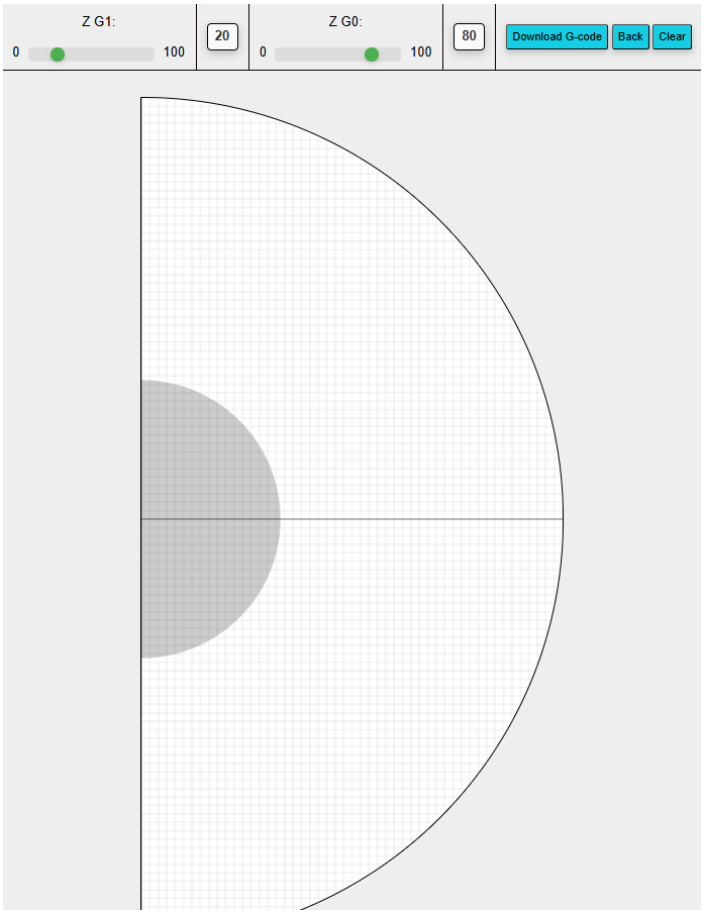
- **استفاده از بخش طراحی**

در این حالت شما وارد محیط جدیدی خواهید شد که با استفاده از طراحی ۲ بعدی می توانید ربات را کنترل نمایید. در ادامه به توضیحات بیشتری درمورد این بخش خواهیم پرداخت.

لازم به ذکر است که با استفاده از دکمه های موجود در این بخش مانند ارسال، اجرا، توقف و ... می‌توانید مانند بخش موقعیت ها، کدها را نیز مدیریت کنید.

۴-۶-۶. بخش طراحی (Paint)

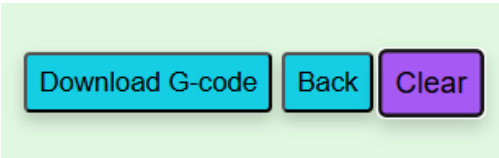
در این بخش، همان‌طور که در شکل ۴-۷۴ مشاهده می‌کنید یک صفحه مختصات به شکل نیم دایره قرار دارد. نقاطی که ربات توانایی حرکت به آنها را دارد، در این نیم‌دایره مشخص شده‌اند. همچنین، با استفاده از اسلایدرهای بالا، می‌توانید ارتفاع بازوی ربات را در نقاط خط‌کشی‌شده (حرکت همراه با انجام کار« Z G1 ») یا مسیرهای آزاد (حرکت سریع ربات « Z G0 ») تنظیم کنید.



شکل ۴- ۷۴- محیط طراحی ۲ بعدی

- **عملکرد دکمه Clear**

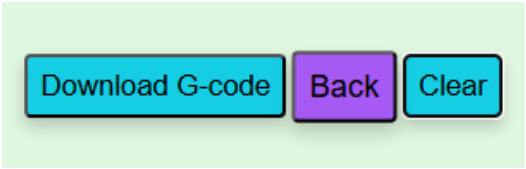
این دکمه به شما امکان می‌دهد تا طرح‌ها و خطوطی که ایجاد کرده‌اید را پاک کنید.



شکل ۴- ۷۵- عملکرد دکمه Clear

عملکرد دکمه Back

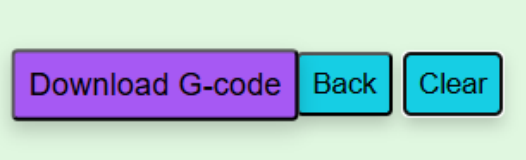
با استفاده از این دکمه، می‌توانید به صفحه قبل یا صفحه اصلی بازگردید.



شکل ۴- ۷۶- عملکرد دکمه Back

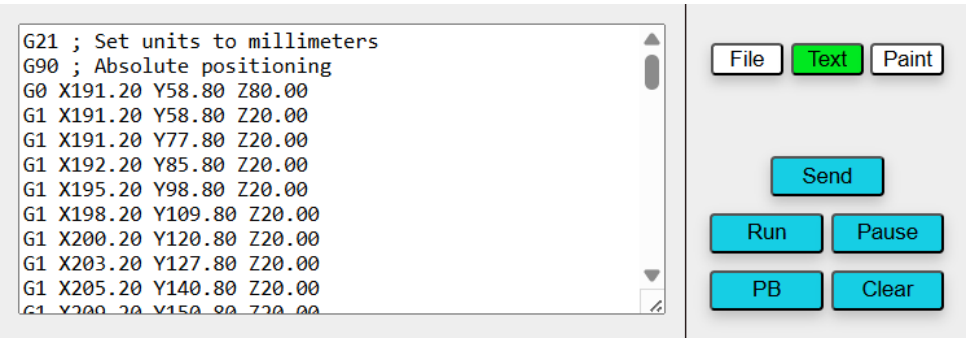
عملکرد دکمه Download Gcode

زمانی که طراحی یا مشخص‌کردن مسیرهای حرکتی روی صفحه به اتمام رسید، می‌توانید از این دکمه استفاده کنید. این دکمه صفحه وب را وادار می‌کند تا یک Gcode متناسب با طرح شما ایجاد کرده و در اختیار شما قرار دهد.



شکل ۴- ۷۷- عملکرد دکمه Download Gcode

همچنین شما می توانید کدهای تولید شده از بخش طراحی را در صفحه اصلی وب مشاهده و در صورت نیاز ویرایش نمایید.



شکل ۴- ۷۸- نمایش کدهای تولید شده در بخش طراحی

۴-۶-۷. بخش نمایش پیام‌های وضعیت

این بخش با نمایش پیام‌های متنی کوتاه، شما را از وضعیت عملکرد ربات مطلع می‌کند. به‌عنوان مثال، زمانی که ربات بر اساس موقعیت‌های تعریف‌شده در حال حرکت باشد، پیام Positions is running نمایش داده می‌شود.

۴-۶-۸. بخش تنظیمات و اطلاعات

این بخش شامل دو بخش اصلی تنظیمات (Setting) و درباره ما (About) می‌باشد.

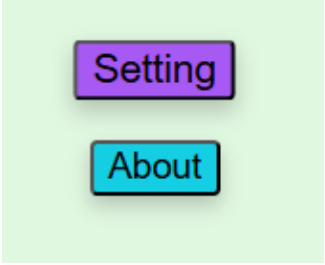
- **بخش درباره ما (About)**

در این بخش، اطلاعاتی مانند نام دانشگاه، اسامی دانشجویان، استاد راهنما، زمان ایجاد پروژه و همچنین لینک دریافت فایل‌های پروژه قرار گرفته است. این اطلاعات به دانشجویانی که علاقه‌مند به ادامه کار و پژوهش در این حوزه هستند، کمک می‌کند تا با ربات و کنترلر طراحی‌شده بیشتر آشنا شوند.

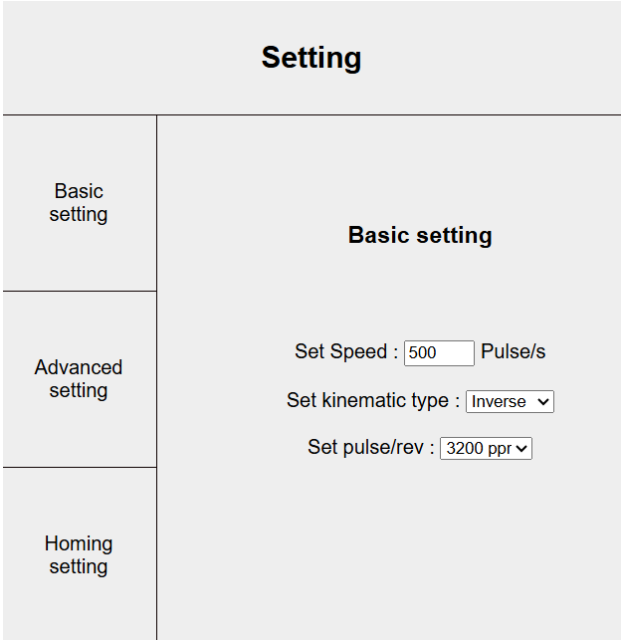
شکل ۴- ۷۹- بخش درباره ما

- **بخش تنظیمات (Setting)**

این قسمت شامل مجموعه‌ای از پارامترهای کنترلی ربات است. کاربران می‌توانند مقادیر تنظیم‌شده را مشاهده کرده یا مقادیر جدیدی برای آن‌ها تعیین کنند. دقت کنید که پارامترهای تنظیمی در این بخش همانند پارامترهای تنظیمی در کنترلر می باشد، بنابراین تغییرات بصورت لحظه ای و هماهنگ در هر دو طرف انجام خواهد شد.



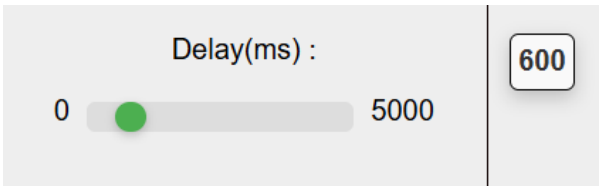
شکل ۴- ۸۰- بخش تنظیمات



شکل ۴- ۸۱- پارامترهای بخش تنظیمات پایه

۴-۶-۹. بخش تنظیم تاخیر در موقعیت ها

تأخیرهای تنظیم‌شده برای موقعیت‌ها به این منظور هستند که ربات در هر نقطه برای مدت زمان مشخصی توقف کند. این تأخیر توسط ماژول ESP8266 اعمال می‌شود. به عبارت دیگر، هنگام ارسال موقعیت‌ها، پس از دریافت پیام از کنترلر مبنی بر اتمام حرکت (یعنی رسیدن ربات به موقعیت فعلی) موقعیت بعدی با تأخیر مشخص‌شده به کنترلر ارسال می‌شود.



شکل ۴- ۸۲- تنظیم فاصله زمانی بین موقعیت ها

۴-۷. بررسی وظایف ماژول ESP8266 با صفحه وب

در این پروژه، ماژول ESP8266 وظیفه دارد که داده‌ها، مختصات، تنظیمات و اطلاعات مختلف را از صفحه وب و کنترلر دریافت کرده و ذخیره نماید. سپس در مواقع ضروری، این داده‌ها را به کنترلر یا صفحه وب ارسال کند. وظایف اصلی این ماژول به شرح زیر است:

۴-۷-۱. ذخیره موقعیت‌های حرکت ربات

یکی از وظایف اصلی ماژول ESP8266 این است که تمامی موقعیت‌هایی که توسط کنترلر یا صفحه وب تنظیم می‌شوند را در حافظه رم[1] خود ذخیره نماید. به محض نیاز، ماژول می‌تواند این موقعیت‌ها را به ترتیب، با تأخیرهای معین و به صورت خط به خط، برای کنترلر ارسال کند.

۴-۷-۱-۱. ویژگی های ذخیره موقعیت ها در حافظه رم

- **افزایش حافظه ذخیره‌سازی کنترلر**

این امر باعث می‌شود که کنترلر بتواند موقعیت‌ها را بدون نیاز به ذخیره‌سازی آنها در حافظه فلش مدیریت کند.

- **دسترسی سریع‌تر و راحت‌تر صفحه وب به موقعیت‌ها**

صفحه وب می‌تواند به‌طور مستقیم موقعیت‌ها و پارامترهای مرتبط با آنها را مشاهده و تغییر دهد.

- **بروز بودن وضعیت سیستم**

وضعیت ماژول ESP8266 و صفحه وب همواره همگام با وضعیت کنترلر و ربات خواهد بود.

۴-۷-۱-۲. چالش های ذخیره موقعیت ها در حافظه رم

- **وابستگی دائمی به ماژول ESP8266**

در صورتی که ارتباط با ماژول ESP8266 قطع شود، کنترلر از دریافت موقعیت‌ها و اطلاعات لازم محروم خواهد شد.

- **تأخیر در ارسال اطلاعات**

اگرچه تأخیر این فرآیند بسیار کم است، اما در موارد خاص ممکن است منجر به تأخیر در حرکت ربات شود.

[1] Random Access Memory (RAM)

۴-۷-۱-۳. علت ذخیره موقعیت‌ها در حافظه رم

ذخیره سازی موقعیت‌ها در حافظه رم ماژول ESP8266 بدلیل سرعت بالای این حافظه در مقایسه با حافظه فلش[1] ضروری است. اگر موقعیت‌ها در حافظه فلش ذخیره شوند، زمان لازم برای خواندن اطلاعات افزایش یافته و این تأخیر می‌تواند موجب مشکلات در حرکت ربات شود. همچنین، بدلیل تغییرات مداوم موقعیت‌ها توسط اپراتور یا سیستم، استفاده از حافظه رم به عنوان محل ذخیره‌سازی، از لحاظ عمر و کارایی برای ماژول مناسب‌تر است، چراکه استفاده مکرر از حافظه فلش می‌تواند باعث کاهش عمر مفید آن شود.

۴-۷-۲. دریافت پارامتر حرکت Move از صفحه وب

زمانی که دستور حرکت Move برای ربات صادر می‌شود، وظیفه ماژول ESP8266 این است که مختصات مربوطه را دریافت و آنها را به فرمت استاندارد تبدیل کند و برای کنترلر ارسال نماید. سپس کنترلر از این مختصات برای حرکت دادن ربات به نقطه مورد نظر استفاده خواهد کرد.

۴-۷-۳. دریافت و پردازش کدهای حرکتی

زمانی که صفحه وب، کدهای حرکتی (Gcode) تولید می‌کند و دکمه ارسال فعال می‌شود، کدهای تولیدشده به‌ترتیب برای ماژول ESP8266 ارسال خواهند شد. وظیفه ماژول این است که این کدها را در متغیرهای مربوطه ذخیره کرده و هنگام نیاز، آن‌ها را به کنترلر ارسال کند. همچنین، هنگامی که دکمه Clear فعال شود، ماژول ESP8266 باید تمامی متغیرهای ذخیره‌کننده کدها را خالی کرده و آماده دریافت و ارسال مجدد کدها گردد.

۴-۷-۴. ارسال موقعیت‌ها و کدهای حرکتی به کنترلر

یکی از وظایف کلیدی ماژول ESP8266 ارسال دقیق و به‌موقع موقعیت‌ها و کدهای حرکتی به کنترلر برای اجرای دستورات حرکتی ربات است. مراحل ارسال موقعیت‌ها به شرح زیر است:

۱) پس از دریافت دستور حرکت، ماژول ESP8266 اولین موقعیت را به کنترلر ارسال می‌کند.
۲) ماژول منتظر می‌ماند تا کنترلر پس از به حرکت درآوردن ربات و رسیدن به موقعیت اول، پیامی مبنی بر اتمام حرکت به ماژول ارسال کند.
۳) پس از دریافت پیام اتمام حرکت، ماژول ESP8266 موقعیت بعدی را به کنترلر ارسال می‌کند و این فرآیند تا ارسال تمامی موقعیت‌ها ادامه پیدا می کند.

[1] FLASH

۴-۷-۵. دریافت تنظیمات از کنترلر به صفحه وب

یکی از چالش‌های موجود در کنترلر این است که پارامترها و مقادیر تنظیمی در کنترلر قابل مشاهده نیستند. به عبارت دیگر، اپراتور تنها می‌تواند تغییرات جدید را اعمال کند، اما نمی‌تواند مقادیر قبلی تنظیم‌شده را مشاهده نماید. برای رفع این مشکل، تمامی پارامترها و مقادیر تنظیمی در کنترلر به ماژول ESP8266 ارسال می‌شود تا ماژول این این پارامترها را برای نمایش در صفحه وب ارسال کند. بدین ترتیب، مقادیر جدید به‌طور دقیق در صفحه وب قابل مشاهده خواهند بود.

۴-۷-۶. ارسال تنظیمات از صفحه وب به کنترلر

یکی از ویژگی‌های مهم این پروژه، امکان تغییر تنظیمات کنترلر از طریق صفحه وب است. به این معنا که علاوه بر مشاهده تنظیمات اعمال‌شده، کاربران قادر به تغییر و اصلاح تنظیمات از طریق صفحه وب خواهند بود. ماژول ESP8266 مسئول دریافت تغییرات تنظیمات اعمال‌شده از صفحه وب و ارسال آن‌ها به کنترلر از طریق ارتباط سریال است تا این تغییرات در تنظیمات کنترلر اعمال شوند.

۴-۸. نحوه ارتباط ماژول ESP8266 با کنترلر

برای برقراری ارتباط بین ماژول ESP8266 و کنترلر، از پروتکل UART استفاده شده که یکی از سریع‌ترین و بهینه‌ترین روش‌های ارتباطی محسوب می‌شود.

۴-۸-۱. ویژگی های استفاده از ارتباط سریال

- **سادگی و کارایی بالا**

ارتباط سریال نیازی به پروتکل‌های پیچیده ندارد و برای انتقال داده‌ها در سیستم‌های نهفته[1] بسیار مناسب است.

- **انعطاف‌پذیری**

ارتباط سریال می‌تواند با سرعت‌های مختلفی کار کند و تنظیمات مختلفی مانند بیت‌های داده، بیت‌های توقف و بیت‌های توازن را پشتیبانی کند.

- **امکان ارسال و دریافت داده‌ها در لحظه**

ارتباط دو طرفه بین کنترلر و ماژول ESP8266 باعث می‌شود که داده‌های حرکتی به صورت بلادرنگ پردازش شوند.

[1] Embedded Systems

۴-۸-۲. پیاده‌سازی ارتباط سریال

در این پروژه، ارتباط سریال به‌صورت دستی[1] طراحی شده است. بدین معنا که داده‌های ارسالی و دریافتی دارای قالب مشخصی هستند که در قالب کدهای سریال تبادل می‌شوند. این کدها بر اساس نیازهای حرکتی ربات طراحی شده‌اند و شامل دستورات کنترلی مانند ذخیره موقعیت‌ها، ارسال فرامین حرکتی و تأیید دریافت داده‌ها می باشند.

جدول ۴- ۲- معرفی بخشی از تبادل کدهای سریال بین کنترلر و ماژول ESP8266

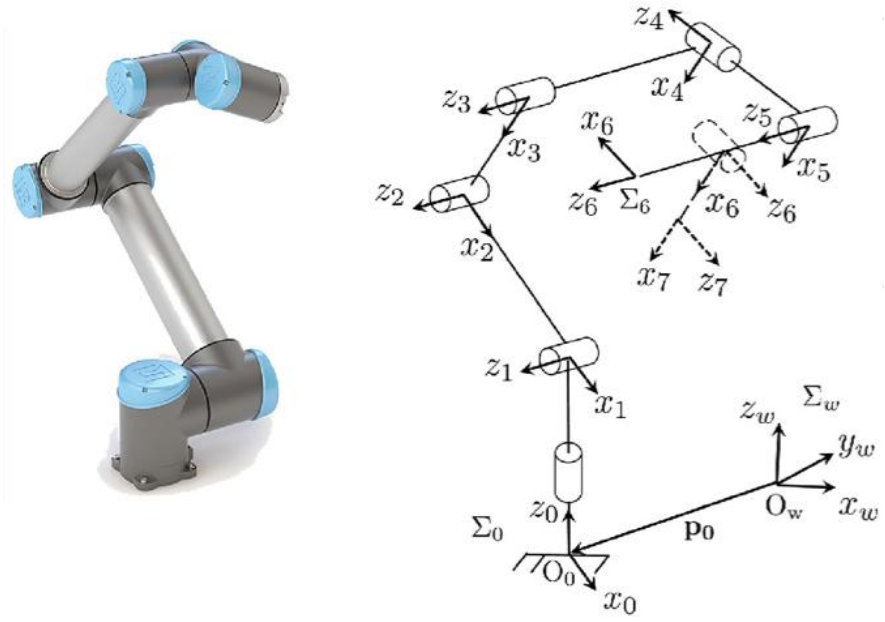
کد سریال	دریافت از طرف	ارسال از طرف	توضیحات
Positions I— = X— Y— Z— T—	ESP	کنترلر	ذخیره موقعیت ها در ماژول ESP8266 و نمایش انها در صفحه وب
Positions I— = X— Y— Z— T—	کنترلر	ESP	ارسال موقعیت ها به کنترلر جهت حرکت ربات
Positions Run	ESP	کنترلر	شروع حرکت
Positions Pause	ESP	کنترلر	توقف موقت حرکت
	کنترلر	ESP	
Positions pBack	ESP	کنترلر	توقف و امکان ادامه حرکت از نقطه اول
	کنترلر	ESP	
Positions Clear	ESP	کنترلر	پاک کردن تمام موقعیت های ذخیره شده
	کنترلر	ESP	
Ok	ESP	کنترلر	تایید دریافت و اتمام حرکت بر اساس اطلاعات ارسال شده

[1] Custom Serial Protocol

فصل پنجم: سینماتیک و پروفایل‌های حرکتی ربات اسکارا

۵-۱. معرفی و بررسی مدلسازی سینماتیکی

سینماتیک در رباتیک علمی است که به بررسی حرکت و موقعیت اجزای ربات‌ها می‌پردازد. این حوزه از ریاضیات و فیزیک، با استفاده از مدلسازی ریاضی، نحوه جابجایی، سرعت و شتاب اجزا را توصیف می‌کند. مدلسازی سینماتیکی ربات به معنای تعریف رابطه‌های ریاضی بین اجزای مکانیکی یک ربات است. برای مدلسازی سینماتیک، از مفاصل و ارتباطات بین قطعات استفاده می‌شود و با توجه به نوع ربات و ساختار آن، از روش‌های مختلفی برای مدلسازی استفاده می‌شود؛ این مدلسازی کمک می‌کند ربات‌ها را برنامه‌ریزی کرد و حرکات دقیق و مورد نیاز برای انجام وظایف مشخص را ایجاد نمود.



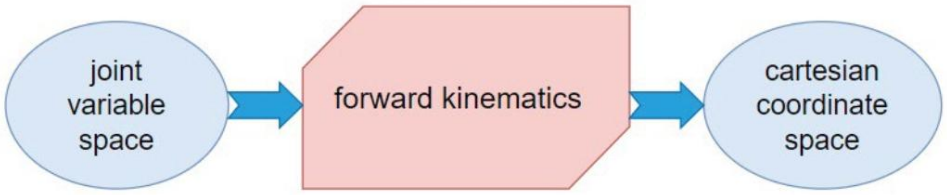
شکل ۵- ۱- مدلسازی یک ربات شش محور

به طور کلی، سینماتیک تکنیکی است که به ما امکان می‌دهد تعیین کنیم چگونه چیزی را از یک موقعیت به موقعیت دیگر منتقل کنیم. وقتی می‌خواهید یک لیوان آب روی میز بگیرید، مغز شما اساساً شکل پیچیده‌ای از سینماتیک را انجام می‌دهد تا بفهمد چگونه مفصل شانه، مفصل آرنج و مچ دست خود را بچرخانید تا دست خود را به سمت لیوان حرکت دهید. سینماتیک ربات به دو دسته، سینماتیک مستقیم[1] و سینماتیک معکوس[2] تقسیم بندی می شود که هرکدام وظایف و نقش‌های مختلفی در فهم و کنترل حرکت ربات‌ها دارند.

[1] Forward Kinematic
[2] Inverse Kinematic

۵-۱-۱. معرفی سینماتیک مستقیم

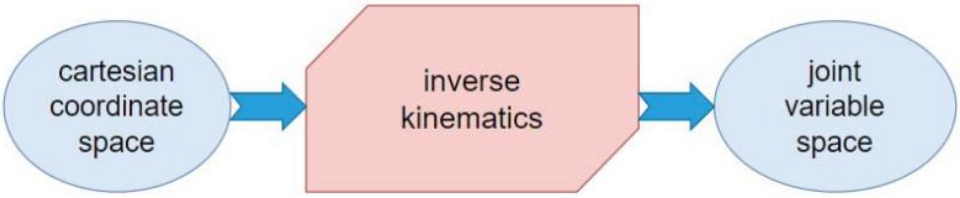
در سینماتیک مستقیم، تلاش برای تعیین حرکت و موقعیت اجزا بر اساس پارامترهای ورودی انجام می‌شود. این پارامترها معمولاً شامل زوایا و ابعاد مکانیکی مانند طول اعضا و اتصالات می‌شوند. با استفاده از این اطلاعات، می‌توان حرکت و موقعیت اجزا را به صورت ریاضی و دقیق مدل کرد. بنابراین هدف از سینماتیک مستقیم، بدست آوردن موقعیت و جهت گیری مجری نهایی[1] با توجه به پیکربندی رابط های میانی ربات است. همانطور که در شکل ۵-۲ مشاهده می کنید، از سینماتیک مستقیم برای تبدیل متغیرهای مفصل به موقعیت مجری نهایی استفاده می‌شود.



شکل ۵- ۲- سینماتیک مستقیم

۵-۱-۲. معرفی سینماتیک معکوس

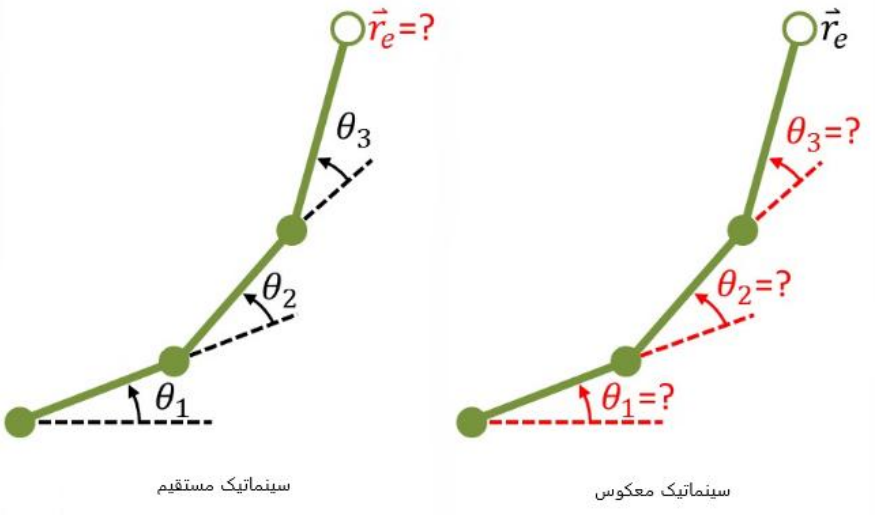
سینماتیک معکوس به معنای تعیین ورودی‌های مورد نیاز برای رسیدن به حرکت یا موقعیت مشخص است. با داشتن حرکت یا موقعیت مورد نظر، این نوع از سینماتیک امکان محاسبه و یافتن ورودی‌های لازم برای دستیابی به آن را فراهم می‌کند. این امر بسیار مهم است زیرا در کنترل ربات‌ها و ماشین‌ها، ممکن است بخواهیم به طور مستقیم یک موقعیت یا حرکت خاص را بدست آوریم و نیاز داریم تا ورودی‌های مورد نیاز برای رسیدن به آن را محاسبه کنیم. بنابراین هدف از سینماتیک معکوس، بدست آوردن موقعیت و جهت گیری رابط های میانی ربات با توجه به موقعیت و جهت گیری مجری نهایی است. همانطور که در شکل ۵-۳ مشاهده می کنید، از سینماتیک معکوس برای یافتن متغیرهای مفصل از موقعیت مجری نهایی استفاده می‌شود.



شکل ۵- ۳- سینماتیک معکوس

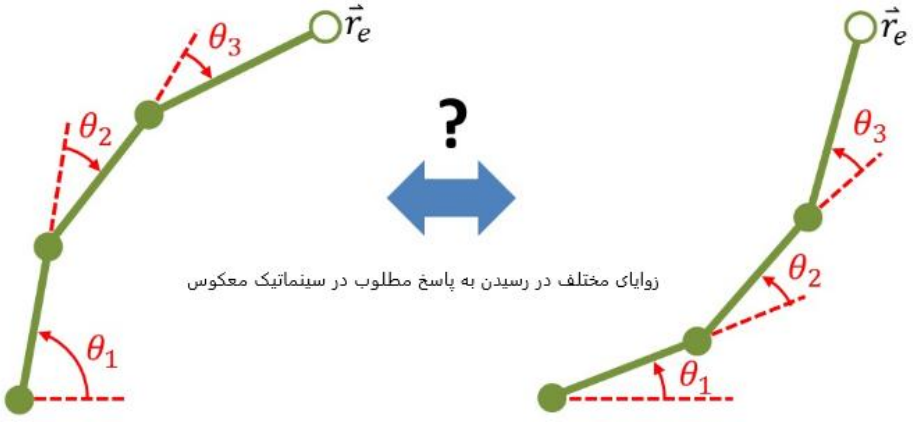
[1] End Effector

حال فرض کنید ما یک بازوی ربات دو بعدی داریم که از چندین مفصل چرخشی تشکیل شده است که به یک مجری نهایی منتهی می شود. عامل پایانی ممکن است یک گیره یا نوعی ابزار باشد. هر مفصل با یک زاویه مشترک تعریف می شود که با θ نشان داده می شود و مجری نهایی در مکانی در فضای فیزیکی قرار دارد که با بردار موقعیت re نشان داده می شود. سینماتیک مستقیم می پرسد، اگر زوایای مفصل را بدانیم، مختصات re چیست؟ از سوی دیگر، سینماتیک معکوس می‌پرسد، اگر ما یک موقعیت مؤثر انتهایی مطلوب داریم که عبارت است از re آنگاه، زوایای مشترک مورد نیاز برای رسیدن به آن موقعیت چیست؟



شکل ۵- ۴- فرم تبدیل سینماتیک مستقیم و معکوس

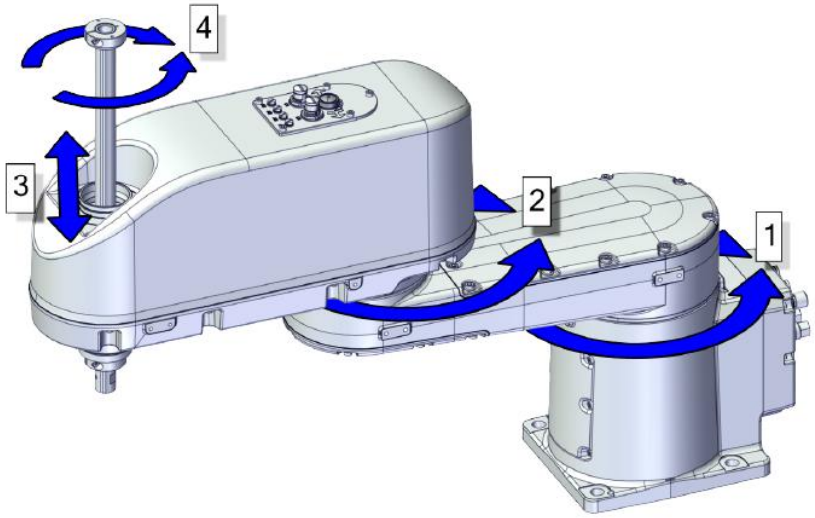
بر خلاف سینماتیک مستقیم، پاسخ به سوال سینماتیک معکوس به دلیل این واقعیت پیچیده است که همیشه یک راه حل واحد ندارد. معمولاً چندین جهت یا موقعیت وجود دارد که به سیستم اجازه می‌دهد به همان موقعیت مجری نهایی هدف برسد. برای بررسی بیشتر این موضوع، ابتدا باید تصمیم بگیریم که چگونه بازوی ربات خود را بصورت ریاضی توصیف کنیم.



شکل ۵- ۵- موقعیت قرارگیری مجری نهایی در سینماتیک معکوس

۵-۱-۳. مدلسازی سینماتیکی ربات اسکارا

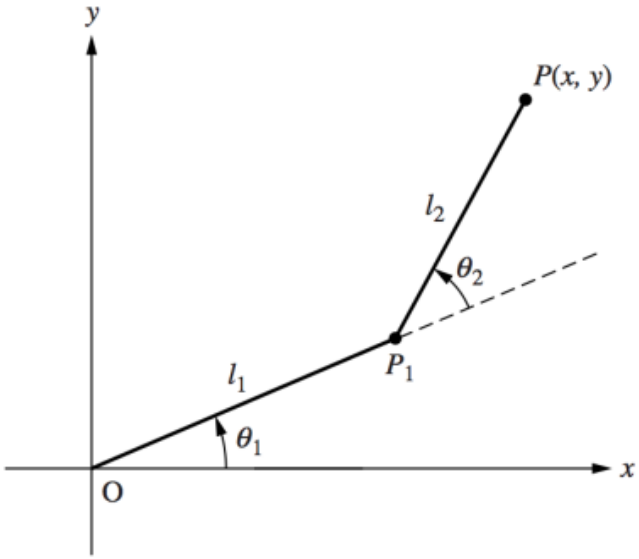
همانطورکه در فصل های پیشین گفته شد، ربات اسکارا سه محور عمودی برای حرکت دورانی دارد (در شکل زیر محورهای ۱ و ۲ و ۴) که از این سه محور یکی حول محور Z و دو محور دیگر نیز بصورت زانویی حول محور Z چرخش انجام می دهند. و درنهایت محور ۳ نیز حرکت خطی در راستای محور Z انجام می دهد.



شکل ۵- ۶- تقسیم بندی محورهای ربات اسکارا

۵-۱-۳-۱. پیاده سازی سینماتیک مستقیم

پیاده سازی سینماتیک مستقیم ربات اسکارا نسبتاً ساده است و به راحتی قابل انجام است. در این بخش، پیاده سازی سینماتیک مستقیم به روش گرافیکی ارائه شده است.



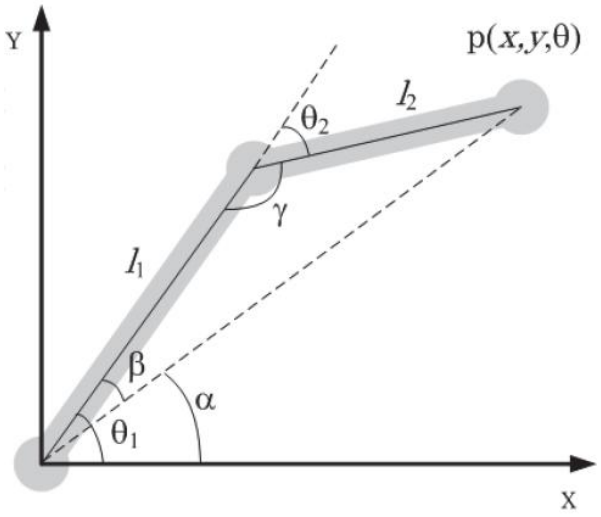
شکل ۵- ۷- پیاده سازی سینماتیک مستقیم ربات اسکارا

در این روش از جبر برداری برای محاسبه مختصات کارتزین ربات اسکارا و موقعیت مجری نهایی استفاده می شود. بدین صورت مختصات X و Y با استفاده از زوایا و طول لینک مفاصل بصورت زیر بدست خواهد آمد.

$$x = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)$$
$$y = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2)$$

۵-۱-۳-۲. پیاده سازی سینماتیک معکوس

به طورکلی، راه‌حل‌های سینماتیک معکوس غیرخطی هستند و پیدا کردن این معادلات می‌تواند پیچیده باشد. در این بخش، پیاده سازی سینماتیک معکوس به روش هندسی ارائه شده است. این روش یکی از ساده ترین راه ها برای پیاده سازی سینماتیک معکوس ربات اسکارا می باشد که میتوان از قانون کسینوس ها استفاده نمود.



شکل ۵- ۸- پیاده سازی سینماتیک معکوس ربات اسکارا

قانون کسینوس ها برای به دست آوردن سه زاویه مثلث در صورت داشتن سه ضلع کاربرد دارد. در واقع قانون کسینوس ها بیان می کند، مربع اندازه هر ضلع برابر است با مجموع مربعات دو ضلع دیگر منهای دو برابر حاصل ضرب آن دو ضلع در کسینوس زاویه بین آنها.

$$c^2 = a^2 + b^2 - 2ab \times \cos(\theta)$$

همانطور که از رابطه بالا نیز معلوم است، با داشتن دو ضلع و زاویه بینشان، می‌توان ضلع مقابل زاویه مذکور را بدست آورد. اما برخلاف این رابطه ما نیازه داریم که زاویه بین اضلاع یا مفاصل ربات را محاسبه نماییم بدین صورت ابتدا قانون کسینوس ها را برای شکل ۵-۸ می نویسیم.

$$x^2 + y^2 = {l_1}^2 + {l_2}^2 - 2l_1 l_2 \cos(180 - \theta_2)$$

حال برای ساده سازی معادله بالا، $\cos(180-\theta_2)$ را برابر با $-\cos(\theta_2)$ قرار می دهیم و بصورت زیر بازنویسی می کنیم.

$$x^2+y^2={l_1}^2+{l_2}^2+2l_1l_2\cos(\theta_2)$$

از آنجایی که ما به زاویه θ_2 نیازمندیم پس معادله فوق را برحسب $\cos(\theta_2)$ بازنویسی می کنیم.

$$\cos(\theta_2)=\frac{x^2+y^2-{l_1}^2-{l_2}^2}{2l_1l_2}$$

حال برای بدست آوردن زاویه θ_2 ، تنها کافیست کسینوس معکوس معادله فوق را محاسبه نماییم.

$$\theta_2=\cos^{-1}\left(\frac{x^2+y^2-{l_1}^2-{l_2}^2}{2l_1l_2}\right)$$

تا اینجای کار ما زاویه θ_2 را از طریق قانون کسینوس ها پیدا کردیم حال با بررسی مجدد شکل ۵-۸ میتوان رابطه زیر را حاصل کرد.

$$\frac{\sin(\beta)}{l_2}=\frac{\sin(\gamma)}{\sqrt{x^2+y^2}}$$

حال برای بدست آوردن زاویه β ، نیاز داریم که معادله فوق را بر حسب $\sin(\beta)$ باز نویسی کنیم و سیسنوس معکوس معادله حاصله را نیز بدست آوریم. از طرفی اگر دقت کنید،$\sin(\gamma)$ برابر است با $\sin(180-\theta_2)$ و همچنین برابر است با $\sin(\theta_2)$ پس با جایگذاری آن، معادله زیر بر حسب زاویه β حاصل خواهد شد.

$$\beta=\sin^{-1}\left(\frac{l_2\sin\theta_2}{\sqrt{x^2+y^2}}\right)$$

زاویه α را نیز میتوان از تانژانت معکوس ضلع مقابل به ضلع مجاور در شکل ۵-۸ بدست آورد.

$$\alpha=\tan^{-1}\left(\frac{y}{x}\right)$$

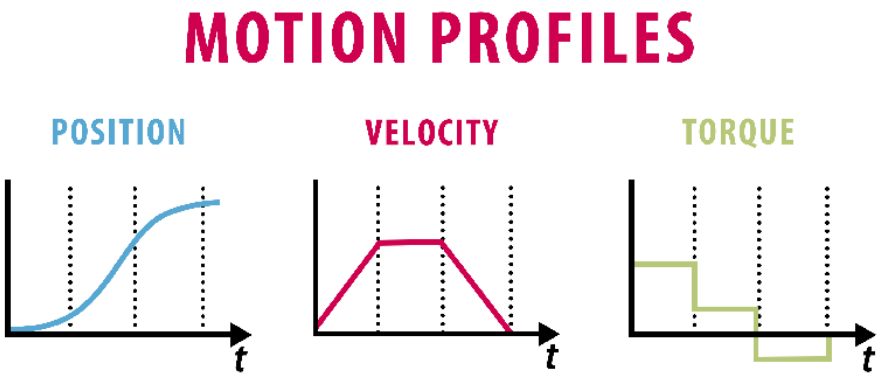
درنهایت زاویه θ_1 با حاصل جمع زوایای β و α بدست می آید و معادله زیر حاصل می شود.

$$\theta_1=\sin^{-1}\left(\frac{l_2\sin\theta_2}{\sqrt{x^2+y^2}}\right)+\tan^{-1}\left(\frac{y}{x}\right)$$

اگر به معادله فوق توجه کنید، مشاهده می کنید که مقدار زاویه θ_1 به مقدار زاویه θ_2 وابسته است بنابراین با تغییر علامت زاویه θ_2 ، دو مقدار متفاوت برای زاویه θ_1 حاصل می شود.

۵-۲. معرفی و بررسی پروفایل های حرکتی

پروفایل‌بندی حرکت، تکنیکی است که به ما امکان می‌دهد تا پارامترهای حرکت ربات را به دقت تنظیم کنیم و بدین ترتیب، حرکات آنرا بهینه و کارآمد کنترل کنیم. این روش موجب می‌شود تا از مشکلاتی مانند لغزش و حرکات ناهموار که می‌تواند به ربات آسیب برساند، جلوگیری شود و حرکت مکانیزم‌های ربات نیز بصورت نرم‌تر و یکنواخت‌تر انجام گیرد. در نتیجه، عملکرد کلی سیستم بهبود یافته و ربات با کارایی بیشتری به وظایف خود می‌پردازد.

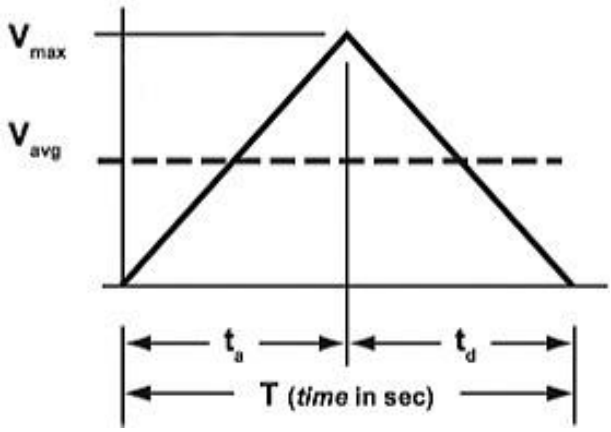


شکل ۵- ۹- معرفی و بررسی پروفایل های حرکتی

یک پروفایل حرکتی اطلاعات حرکت فیزیکی را ارائه می‌دهد و به صورت گرافیکی نشان می‌دهد که موتور باید در حین حرکت از نظر موقعیت، سرعت و شتاب چگونه رفتار کند یا به زبان ساده یک پروفایل حرکتی، نحوه رفتار موتور را از نظر موقعیت، سرعت و شتاب در طول حرکت تعریف می‌کند. اگرچه پروفایل‌های حرکتی بسیاری وجود دارد که می‌توانند یک حرکت فیزیکی خاص را تحقق بخشند، اما دو نوع رایج‌ترین پروفایل حرکت، مثلثی و ذوزنقه‌ای هستند. این نام‌ها به دلیل شکلی که وقتی سرعت به عنوان تابعی از زمان ترسیم و نمایش داده می‌شود، انتخاب شده‌اند.

۵-۲-۱. پروفایل حرکت مثلثی

اصل اساسی پروفایل حرکت مثلثی این است که با حداکثر سرعت شتاب بگیریم و سپس بلافاصله کاهش سرعت دهیم، با این تفاوت که شتاب و کاهش سرعت از نظر زمان و مسافت برابر هستند. به عبارتی، زمان حرکت به دو قسمت مساوی تقسیم می‌شود. در نیمه اول، موتور شتاب می‌گیرد تا به حداکثر سرعت برسد و در نیمه دوم، تا زمانیکه موتور توقف کند سرعت کاهش می‌یابد. این تقسیم زمانی به طراحان این امکان را می‌دهد تا به راحتی شتاب و سرعت حرکت را محاسبه کنند. این باعث می‌شود که محاسبات سرعت و شتاب بر اساس هندسه یک مثلث ساده‌تر شوند.

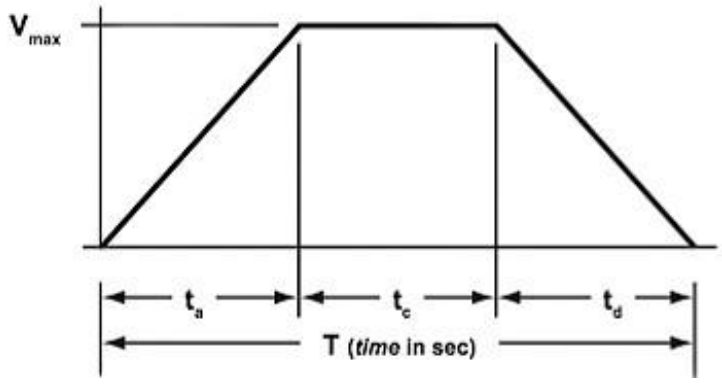


شکل ۵- ۱۰- پروفایل حرکت مثلثی

همانطورکه در شکل ۵-۱۰ مشاهده می کنید، ارتفاع مثلث نشان‌دهنده سرعت حداکثر است و شتاب با تقسیم سرعت حداکثر بر زمان شتاب، که نصف زمان کل حرکت است، به دست می‌آید. پروفایل حرکت مثلثی به عنوان سریع‌ترین گزینه برای حرکت از یک موقعیت به موقعیت دیگر است، این ویژگی باعث می‌شود پروفایل حرکت مثلثی برای کاربردهایی که نیاز به حداقل زمان برای حرکت دارند مناسب باشد. همچنین این پروفایل بطور معمول برای کاربردهایی استفاده می‌شود که نیاز به سرعت ثابت ندارند مانند حمل و نقل، جابجایی و جاگذاری زیرا سریع‌ترین حرکت بین دو موقعیت را فراهم می‌کنند.

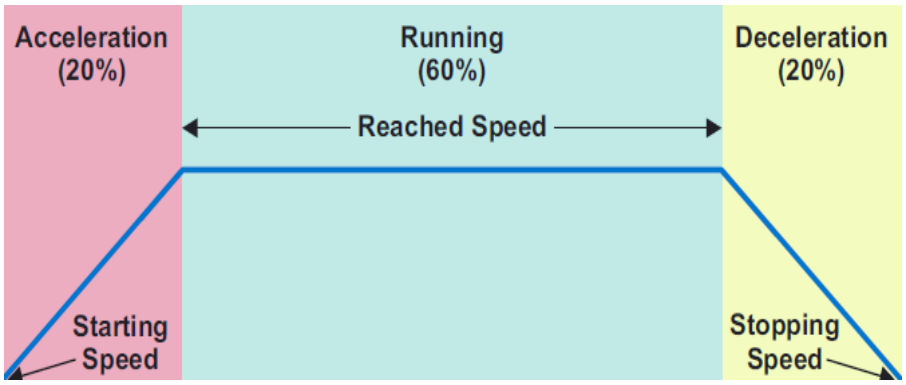
۵-۲-۲. پروفایل حرکت ذوزنقه‌ای

برخلاف پروفایل حرکت مثلثی، پروفایل حرکت ذوزنقه‌ای زمان مشخصی (یا فاصله) را برای حرکت با سرعت ثابت اختصاص می‌دهد که باعث می شود محاسبات حرکت برای پروفایل حرکت ذوزنقه‌ای کمی پیچیده‌تر از پروفایل حرکت مثلثی باشد. بهترین روش برای تحلیل یک پروفایل حرکت ذوزنقه‌ای، تقسیم آن به دو مثلث (برای دوره‌های شتاب و کاهش سرعت حرکت) و یک مستطیل (برای دوره سرعت ثابت) است.



شکل ۵- ۱۱- پروفایل حرکت ذوزنقه‌ای

ساده‌ترین روش برای ایجاد پروفایل حرکت ذوزنقه‌ای، همانند شکل ۵-۱۱ تقسیم کل زمان حرکت به سه قسمت مساوی است، بدین شکل در یک‌سوم اول، موتور شتاب می‌گیرد تا به حداکثر سرعت حرکت برسد و در یک‌سوم دوم، موتور با سرعت ثابت حرکت می‌کند و در یک‌سوم آخر، تا زمانیکه موتور توقف کند سرعت کاهش می‌یابد. این تقسیم زمانی به طراحان این امکان را می‌دهد تا به راحتی شتاب و سرعت حرکت را محاسبه کنند. اما معمولاً پروفایل حرکت ذوزنقه‌ای از مدت زمان بیشتری برای سرعت ثابت استفاده می‌کند و نرخ‌های شتاب و کاهش سرعت کمتری نیز دارد. به همین دلیل، روش دیگری برای ایجاد پروفایل حرکت ذوزنقه‌ای وجود دارد که کل زمان حرکت را به درصدهایی از هر منطقه خاص پروفایل حرکت ذوزنقه‌ای تقسیم می کند. بدین شکل می‌توان ۲۰ درصد از کل زمان را برای شتاب‌گیری موتور، ۶۰ درصد را برای حرکت موتور با سرعت ثابت و ۲۰ درصد باقی‌مانده را برای کاهش سرعت موتور انتخاب کرد.



شکل ۵- ۱۲- روش دوم برای ایجاد پروفایل حرکت ذوزنقه‌ای

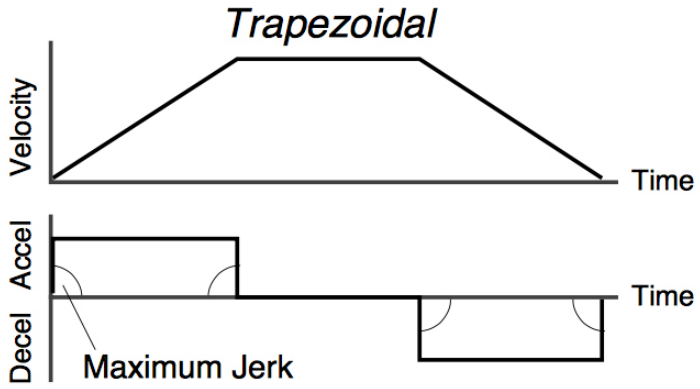
پروفایل حرکت ذوزنقه‌ای شاید متداول‌ترین نوع پروفایل در کاربردهای کنترل حرکت باشد، زیرا برای کاربردهایی که از یک دوره سرعت ثابت بهره‌مند می‌شوند، یک پروفایل حرکت ذوزنقه‌ای مزیت داشتن حداکثر سرعت کمتر نسبت به پروفایل حرکت مثلثی را ارائه می‌دهد. این ویژگی می‌تواند به کاهش فشار و سایش روی سیستم کمک کند و برای کاربردهایی مانند ماشین‌کاری، توزیع، چاپ و رنگ‌آمیزی که نیاز به حفظ سرعت ثابت برای مدت زمان مشخص دارند، بسیار مفید باشد.

۵-۲-۳. پروفایل حرکت S-Curve

در واقعیت، هیچ‌کدام یک از پروفایل‌های حرکتی که به توضیح آنها پرداختیم برای سیستم‌های حرکتی به ویژه سیستم‌هایی که به حرکت صاف، دقت بالای موقعیت‌یابی یا پایداری در پایان حرکت نیاز دارند، مناسب نیستند. این به این دلیل است که فرآیند شتاب‌گیری و کاهش سرعت منجر به پدیده‌ای به نام جهش[1] می‌شود.

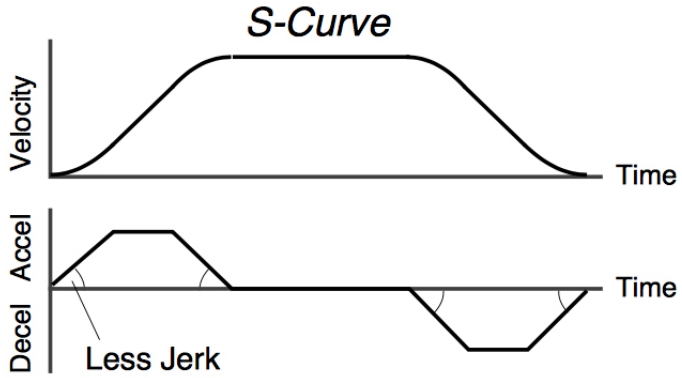
[1] Jerk

همانطورکه سرعت مشتق جابجایی است و شتاب مشتق سرعت، جهش نیز مشتق شتاب است. به عبارتی جهش، نرخ افزایش یا کاهش شتاب است. جهش بطور کلی نامطلوب است زیرا باعث ایجاد حرکات ناگهانی و ناپایدار می شود. در کاربردهای صنعتی مانند ماشین ابزار، ربات اسکارا و سیستم‌های پخش‌کننده، تغییر سریع در شتاب (یعنی جهش) باعث لرزش سیستم می‌شود. هرچه جهش بیشتر باشد، لرزش نیز بیشتر است و این لرزش‌ها، دقت موقعیت‌یابی را کاهش و زمان نشست[1] را افزایش می‌دهند.



شکل ۵- ۱۳- پروفایل حرکت ذوزنقه‌ای با حداکثر نرخ جهش

برای اجتناب از جهش، باید نرخ افزایش یا کاهش شتاب را کاهش دهیم. در سیستم‌های کنترل حرکت، اینکار با استفاده از پروفایل حرکت S-curve بجای پروفایل حرکت ذوزنقه‌ای انجام می‌شود. همانطور که در شکل ۵-۱۳ مشاهده می کنید در یک پروفایل حرکت ذوزنقه‌ای، شتاب‌گیری بلافاصله اتفاق می‌افتد و نرخ جهش حداکثر است. برای کاهش میزان جهشی که در حین حرکت ایجاد می‌شود، تغییرات سرعت در ابتدا و انتهای شتاب‌گیری به شکل S هموار می‌شوند. بدین صورت پروفایل حاصله به عنوان پروفایل حرکت S-curve شناخته می‌شود.

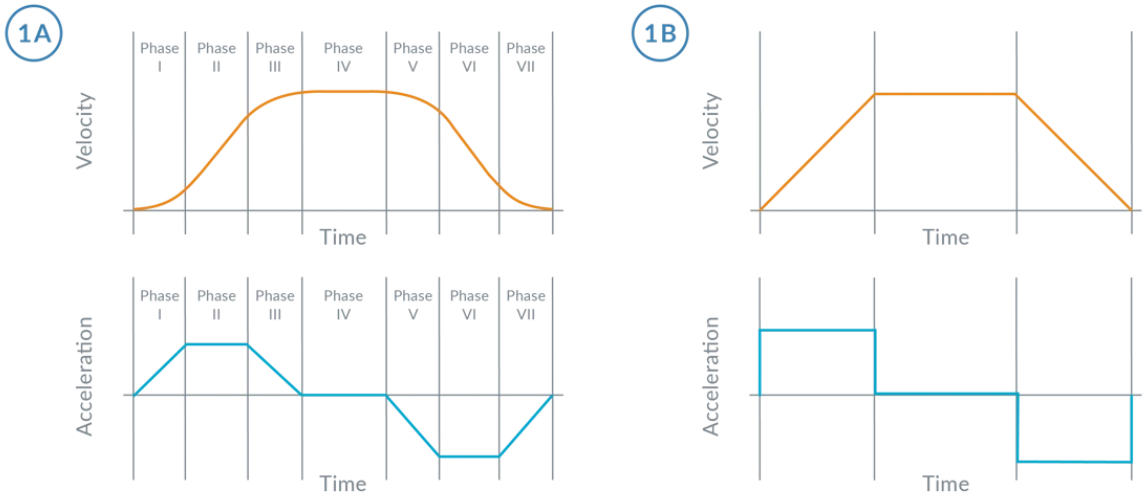


شکل ۵- ۱۴- پروفایل حرکت S-curve با حداقل نرخ جهش

[1] Settling time

۵-۲-۴. مقایسه پروفایل حرکت S-Curve و ذوزنقه‌ای

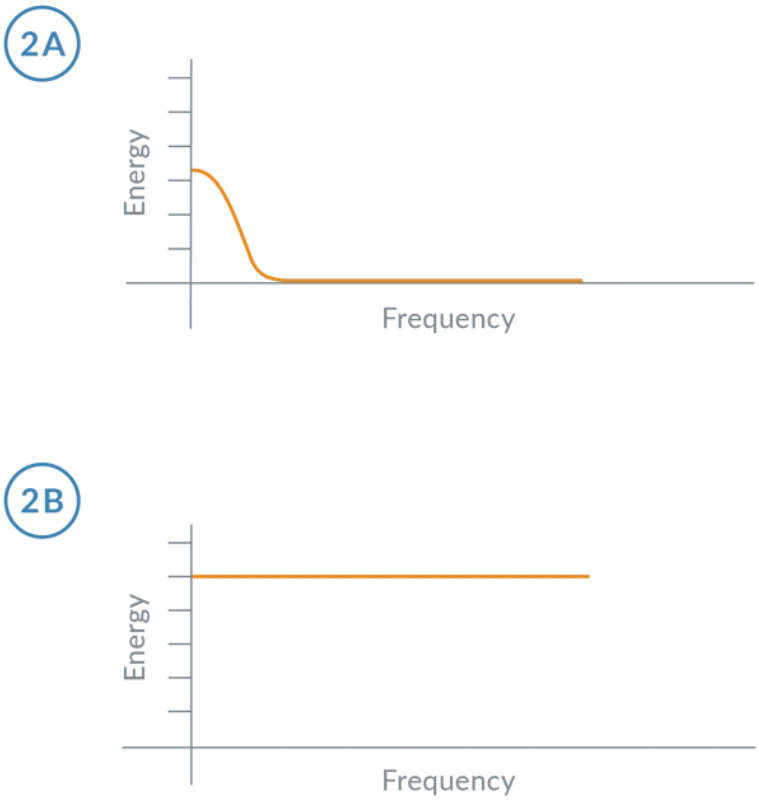
پروفایل حرکت S-curve بر اساس یک سیستم مرتبه سوم ایجاد می شود که باعث پیچیده‌تر شدن معادلات حرکت برای شتاب، سرعت و مسافت (جابجایی) نسبت به پروفیل حرکت ذوزنقه‌ای می‌شود. استفاده از پروفایل حرکت S-curve در مقایسه با پروفایل حرکت ذوزنقه‌ای باعث طولانی‌تر شدن زمان کلی حرکت می‌شود. دلیل این امر آن است که زمان شتاب‌گیری و کاهش سرعت در پروفیل حرکت S-curve بیشتر از شتاب‌گیری لحظه‌ای در پروفایل حرکت ذوزنقه‌ای طول می‌کشد. با این حال، مزیت زمانی که با استفاده از پروفایل حرکت ذوزنقه‌ای به دست می‌آید، ممکن است به دلیل طولانی‌تر شدن زمان نشست به علت لرزش‌های ناشی از سطوح بالای جهش خنثی شود و به دلیل اینکه جهش فشار زیادی به اجزای مکانیکی وارد می‌کند، حتی اگر از پروفایل حرکت ذوزنقه‌ای استفاده شود، معمولاً مقداری صاف‌سازی در مراحل شتاب‌گیری و کاهش سرعت اعمال می‌شود، که باعث می‌شود پروفایل حرکت به شکل S باشد.



شکل ۵- ۱۵- مقایسه پروفایل حرکت S-Curve و ذوزنقه‌ای

همانطورکه در شکل ۵-۱۵ بخش 1A مشاهده می نمایید، یک پروفایل حرکت S-curve شامل ۷ مرحله متمایز از حرکت است. از طرفی، یک پروفایل حرکت ذوزنقه‌ای دارای ۳ مرحله است که فقط دارای مراحل مربوط به شماره ۲ (شتاب ثابت)، شماره ۴ (سرعت ثابت) و شماره ۶ (کاهش سرعت ثابت) از پروفایل S-curve است. در یک پروفایل حرکت ذوزنقه‌ای، نرخ جهش (تغییر شتاب) حداکثر است، در حالیکه در یک پروفایل حرکت S-curve، جهش یک مقدار ثابت است و تغییرات شتاب را در طول یک دوره زمانی پخش می‌کند. اما از آنجایی که پروفایل حرکت ذوزنقه‌ای زمان خود را صرف شتاب کامل یا کاهش سرعت کامل می‌کند، از دیدگاه اجرای پروفایل حرکت، سریع‌تر از پروفایل حرکت S-curve عمل می کند.

اینکه یک پروفایل حرکت S-curve نسبت به یک پروفایل حرکت ذوزنقه‌ای نرم‌تر است، از منحنی های بالا مشخص است. اما چرا پروفایل حرکت S-curve باعث کاهش نوسانات بار می‌شود؟ پاسخ به این سوال به این واقعیت برمی‌گردد که برای یک بار معین، هرچه جهش بیشتر باشد، مقدار انرژی ارتعاشی ناخواسته بیشتری تولید می‌شود و طیف فرکانسی گسترده‌تری از انرژی ارتعاشی ایجاد می‌شود. این بدان معناست که هرچه تغییر شتاب سریع‌تر باشد، ارتعاشات قوی‌تر خواهند بود و از آنجاییکه انرژی ارتعاشی در مکانیک سیستم جذب می‌شود، ممکن است باعث افزایش زمان نشست یا کاهش دقت شود.

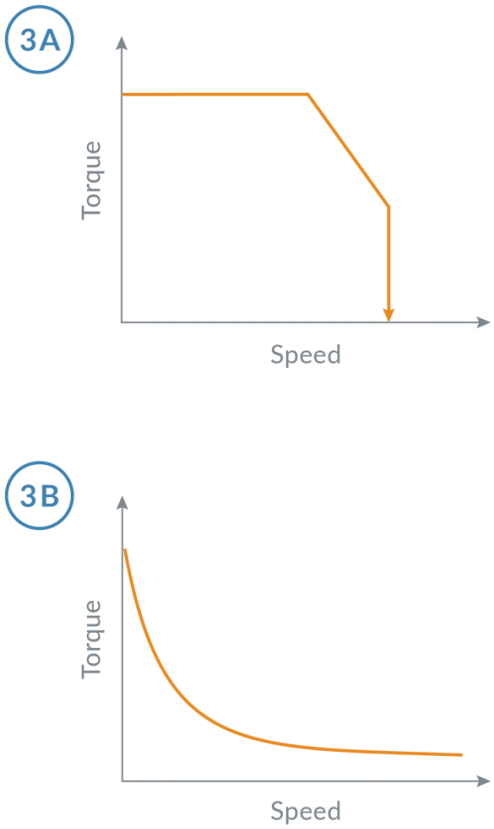


شکل ۵- ۱۶- ارتعاشات القایی برای پروفایل حرکت S-curve (2A) و پروفایل حرکت ذوزنقه‌ای (2B)

همانطورکه در شکل ۵-۱۶ مشاهده می کنید، پروفایل حرکت S-curve به دلیل داشتن جهش یکنواخت و تغییرات نرم‌تر در شتاب، ارتعاشات کمتری القا می‌کند در حالی که پروفایل حرکت ذوزنقه‌ای به دلیل تغییرات آنی بین مراحل شتاب و کاهش سرعت، ارتعاشات بیشتری ایجاد می‌کند. این تفاوت در ارتعاشات القایی به ما کمک می‌کند تا بفهمیم چرا پروفایل حرکت S-curve برای کاربردهایی که نیاز به حرکات نرم‌تر و کاهش ارتعاشات دارند، مناسب‌تر است. انتخاب پروفایل حرکت مناسب، بستگی به نیازها و مشخصات عملکردی سیستم شما دارد.

۵-۲-۵. چالش های پروفایل‌های حرکتی

هدف نهایی هر پروفایل حرکتی، تطبیق ویژگی‌های سیستم حرکت با کاربرد مورد نظر است. پروفایل حرکت S-curve و ذوزنقه‌ای زمانیکه منحنی گشتاور دور سیستم حرکت به طور نسبی ثابت است، به خوبی کار می‌کنند به عبارتی، گشتاور خروجی در طول محدوده سرعت‌هایی که سیستم تجربه خواهد کرد، تغییر زیادی نمی‌کند. این برای اکثر سروو موتورها صادق است. اما استپر موتورها منحنی گشتاور دور ثابتی ندارند و خروجی گشتاور آنها غیرخطی است و گاهی اوقات در مکانی به نام ناپایداری میانی[1] کاهش زیادی دارند و بطور کلی در سرعت‌های بالاتر نیز کاهش می‌یابد.

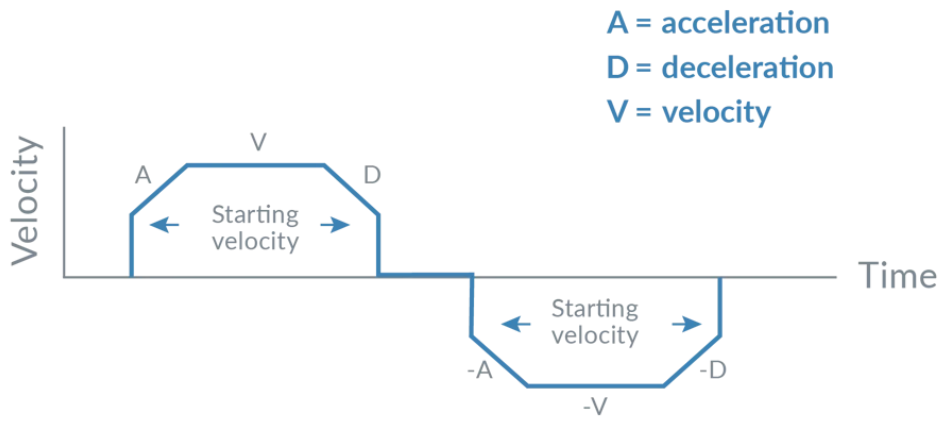


شکل ۵- ۱۷- منحنی گشتاور دور سروو موتور (3A) و استپرموتور (3B)

ناپایداری میانی در فرکانس گام[2] زمانی رخ می‌دهد که فرکانس رزونانس طبیعی موتور با نرخ گام فعلی مطابقت داشته باشد. در این حالت، ارتعاشات و نوسانات ناخواسته‌ای در سیستم ایجاد می‌شود که منجر به کاهش دقت و عملکرد موتور می‌گردد. فرکانس گام به تعداد پله‌هایی اشاره دارد که یک استپرموتور در واحد زمان برمی‌دارد. به عبارتی، فرکانس گام نشان می‌دهد که استپرموتور با چه سرعتی می‌تواند بین موقعیت‌های مختلف جابجا شود. فرکانس گام معمولاً بر حسب هرتز (Hz) اندازه‌گیری می‌شود.

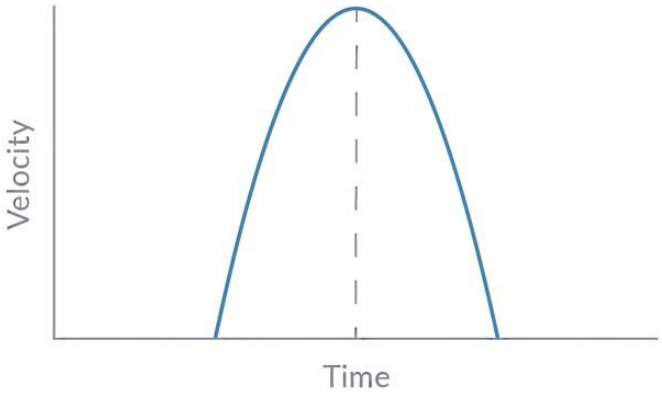
[1] Mid-range Instability
[2] Step frequency

برای رفع ناپایداری میانی، تکنیک‌های مختلفی وجود دارد. یکی از رایج‌ترین تکنیک‌ها استفاده از سرعت اولیه غیرصفر است، که به معنی شروع به حرکت با سرعتی مشخص بجای شروع از حالت ایستاده است. اینکار می‌تواند به کاهش تأثیرات ناپایداری میانی کمک کند و باعث بهبود عملکرد سیستم شود. اگرچه این روش نسبتاً غیر استاندارد است، اما گاهی اوقات نتایج بهتری نسبت به شیب صاف برای سرعت اولیه صفر ارائه می‌دهد.



شکل ۵- ۱۸- منحنی سرعت اولیه غیرصفر

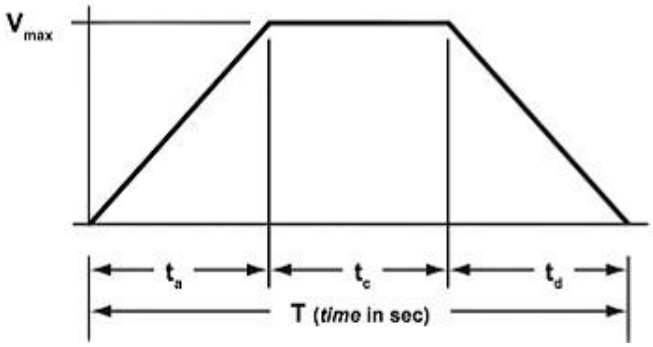
برای مقابله با کاهش گشتاور در سرعت‌های بالاتر نیز می‌توان از پروفایل حرکت پارابولیک استفاده کرد که در شکل ۵-۱۹ نشان داده شده است. این پروفایل حرکتی با استپر موتورها سازگاری خوبی دارد، زیرا گشتاور در سرعت‌های بالاتر کمتر است. اما توجه داشته باشید که شتاب‌گیری شروع و پایان بسیار بالا است و هیچ یک از مراحل پروفایل حرکت S-curve که در آن شتاب بطور نرم به صفر انتقال یابد وجود ندارد. بنابراین، اگر نوسانات بار مشکل اصلی باشد، پروفایل حرکت پارابولیک ممکن است به خوبی پروفایل حرکت S-curve عمل نکند، علیرغم اینکه یک پروفایل حرکت S-curve نیز از دیدگاه منحنی گشتاور دور برای یک استپرموتور بهینه نشده است.



شکل ۵- ۱۹- پروفایل حرکت پارابولیک

۵-۲-۶. پیاده سازی پروفایل حرکت ذوزنقه‌ای برای استپرموتور

همانطورکه قبلا گفتیم بهترین روش برای تحلیل یک پروفایل حرکت ذوزنقه‌ای، تقسیم آن به دو مثلث (برای دوره‌های شتاب و کاهش سرعت حرکت) و یک مستطیل (برای دوره سرعت ثابت) است. بدین صورت ما با دانستن فرمول های مساحت مثلث و مستطیل می توانیم پارامترهای سرعت، شتاب و مسافت را محاسبه نماییم.



شکل ۵- ۲۰- پروفایل حرکت ذوزنقه‌ای

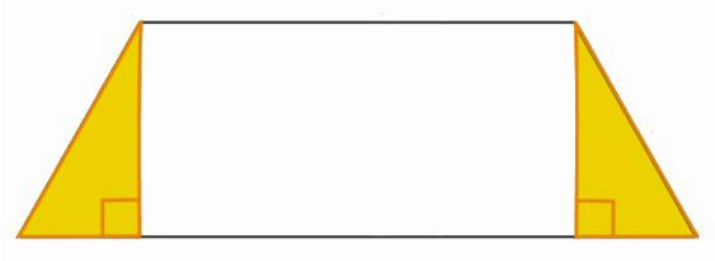
برای شروع، ابتدا نیاز داریم پارامترهای کلی پروفایل حرکت ذوزنقه‌ای را بدست آوریم. بدین صورت طبق شکل ۵-۲۰ ، زمان کل بصورت زیر محاسبه می شود.

$$T = t_a + t_c + t_d$$

با توجه به اینکه زمان کل از جمع زمان های بخش شتاب، سرعت ثابت و کاهش سرعت حاصل می شود، میتوان نتیجه گرفت که مسافت کل نیز به سه قسمت تقسیم می شود و بصورت زیر محاسبه می شود.

$$D = d_a + d_c + d_d$$

حال برای محاسبه مسافت سه بخش شتاب، سرعت ثابت و کاهش سرعت، باید همانطورکه گفتیم مساحت مثلث (برای دوره‌های شتاب و کاهش سرعت حرکت) و مساحت مستطیل (برای دوره سرعت ثابت) بدست آوریم. مساحت مثلث برابر است با قاعده ضربدر ارتفاع تقسیم بر دو و مساحت مستطیل نیز برابر است با طول ضرب در عرض.



شکل ۵- ۲۱- تقسیم پروفایل حرکت ذوزنقه‌ای به دو مثلث و یک مستطیل

با محاسبه مساحت مثلث و مستطیل طبق شکل ۵-۲۱ ، مسافت هر بخش بصورت زیر بدست خواهد آمد.

$$d_a = d_d = \frac{1}{2} \times b \times h \quad , \quad d_c = A \times B$$

$$d_a = \frac{1}{2} \times t_a \times V_{max} \quad , \quad d_c = t_c \times V_{max} \quad , \quad d_d = \frac{1}{2} \times t_d \times V_{max}$$

مسافت کل نیز بصورت زیر محاسبه می شود.

$$D = \left(\frac{1}{2} \times t_a \times V_{max}\right) + (t_c \times V_{max}) + \left(\frac{1}{2} \times t_d \times V_{max}\right)$$

خب حالا برای اینکه بتوانیم تشخیص دهیم کدام نقطه از پروفایل، نقطه انتهای شتاب گیری می باشد، باید مسافت بخش شتاب را محاسبه نماییم.

$$d_a = \frac{1}{2} \times t_a \times V_{max}$$

همانطورکه در معادله فوق مشاهده می کنید، برای محاسبه مسافت بخش شتاب باید پارامتر t_a یا زمان شتاب گیری را بدانیم. همانطورکه می دانید به میزان تغییر سرعت در واحد زمان، شتاب گفته می‌شود، پس بدین صورت می توانیم پارامتر t_a را از فرمول شتاب بدست می آوریم.

$$a = \frac{\Delta v}{t_a} = \frac{v_2 - v_1}{t_a} \rightarrow v_2 = V_{max} \ , \ v_1 = 0 \rightarrow a = \frac{V_{max}}{t_a} \rightarrow t_a = \frac{V_{max}}{a}$$

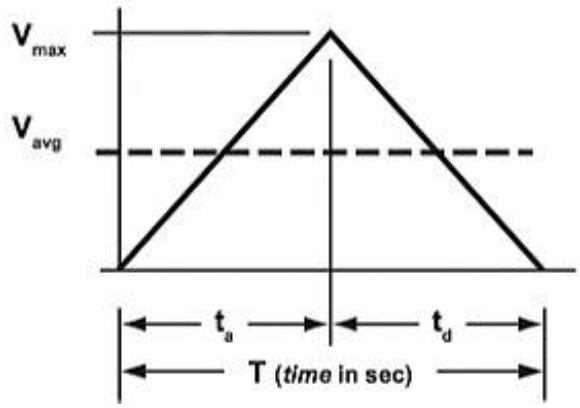
سپس با جایگذاری پارامتر t_a در معادله مسافت بخش شتاب، فرمول زیر حاصل می شود.

$$d_a = \frac{1}{2} \times \left(\frac{V_{max}}{a}\right) \times V_{max} = \frac{(V_{max})^2}{2a}$$

حال طبق فرمول فوق، می توانیم نقطه انتهای شتاب گیری را تعیین نماییم و از این نقطه به بعد وارد بخش سرعت ثابت خواهیم شد. همانند نقطه انتهای شتاب گیری، ما نیاز به نقطه شروع کاهش سرعت یا حرکت با شتاب منفی را نیز داریم که این نقطه بعد از به اتمام رسیدن بخش سرعت ثابت شروع می شود. برای اینکار باید مسافت کل را از مسافت بخش شتاب کم کنیم تا نقطه انتهای بخش سرعت ثابت یا نقطه شروع کاهش سرعت بدست آید.

$$d_c = D - d_a$$

خب حالا با توجه به تعاریفی که انجام شد ما دو سناریو متفاوت خواهیم داشت، اولین سناریو این است که اگر d_c بزرگتر از d_a باشد بدین معناست که پروفایل حرکت ذوزنقه‌ای ایجاد خواهد شد و بخش سرعت ثابت در این پروفایل وجود خواهد داشت. دومین سناریو این است که اگر d_c کوچکتر از d_a باشد بدین معناست که پروفایل حرکت ذوزنقه‌ای ایجاد نخواهد شد و بخش سرعت ثابت نیز در این پروفایل وجود نخواهد داشت پس پروفایل حرکت ایجاد شده، پروفایل حرکت مثلثی خواهد بود و در این حالت باید محاسبات جدیدی بدست آید.



شکل ۵- ۲۲- پروفایل حرکت مثلثی

طبق شکل ۵-۲۲ ، زمان کل برابر است با حاصل جمع زمان بخش شتاب و زمان بخش کاهش سرعت. پس برای محاسبه زمان هر یک از بخش های شتاب و کاهش سرعت، فقط نیاز داریم که کل زمان رو بر دو تقسیم کنیم.

$$T = t_a + t_d \;\rightarrow\; t_a = \frac{T}{2} \;\;,\;\; t_d = \frac{T}{2}$$

در پروفایل حرکت مثلثی بر خلاف پروفایل حرکت ذوزنقه‌ای سرعت حداکثر برای ما مجهول است و نیاز داریم از طریق فرمول سرعت متوسط، سرعت حداکثر را محاسبه نماییم.

$$V_{avg} = \frac{V_{max}}{2} \;\rightarrow\; V_{max} = 2 \times V_{avg}$$

سرعت متوسط نیز از حاصل تقسیم مسافت کل به زمان کل بدست می آید، پس برای محاسبه سرعت متوسط ابتدا نیاز داریم زمان کل را بدست آوریم. همچنین زمان کل نیز از فرمول شتاب محاسبه می شود.

$$a = \frac{V_{max}}{t_a} \;\;,\;\; t_a = \frac{T}{2} \rightarrow a = \frac{V_{max}}{\frac{T}{2}} = \frac{2 \times V_{max}}{T} \rightarrow T = \frac{2 \times V_{max}}{a}$$

حال بصورت زیر، سرعت حداکثر را از طریق فرمول سرعت متوسط محاسبه می نماییم.

$$V_{avg} = \frac{D}{T} = \frac{D}{\frac{2 \times V_{max}}{a}} = \frac{a \times D}{2 \times V_{max}} \rightarrow V_{max} = 2 \times V_{avg} = \frac{a \times D}{V_{max}} \rightarrow (V_{max})^2 = a \times D$$

$$V_{max} = \sqrt{a \times D}$$

از طرفی نقاط انتهای شتاب گیری و ابتدای کاهش سرعت نیز با تقسیم مسافت کل بر دو مجددا محاسبه خواهد شد. این نقاط همان مسافت بخش شتاب و بخش کاهش سرعت پروفایل حرکت مثلثی می باشند.

$$d_a = \frac{D}{2} \quad , \quad d_c = \frac{D}{2}$$

خب تا اینجای کار ما مسافت بخش های شتاب، سرعت ثابت و کاهش سرعت یک پروفایل حرکت ذوزنقه‌ای را محاسبه کردیم اما اگر متوجه شده باشید این پروفایل برای استپرموتور پیاده سازی شده بود پس مسافت های هر بخش برحسب تعداد گام ها یا پله های استپرموتور محاسبه خواهد شد. حالا در ادامه وارد قسمت مهم کار می شویم، در اینجا ما نیاز داریم که سرعت را برای هر پله محاسبه نماییم بدین صورت سرعت لحظه‌ای باید برای هر بخش شتاب، سرعت ثابت و کاهش سرعت بدست آید. سرعت لحظه‌ای یا سرعت هر پله برای بخش شتاب بصورت زیر محاسبه می شود.

$$d_a = \frac{(V_{max})^2}{2a} \rightarrow (V_{max})^2 = d_a \times 2a \rightarrow V_{max} = \sqrt{d_a \times 2a} \rightarrow V_{accel} = \sqrt{d_a \times 2a}$$

همچنین سرعت لحظه‌ای برای بخش سرعت ثابت نیز با سرعت حداکثر برابر است.

$$V_{const} = V_{max}$$

سرعت لحظه‌ای برای بخش کاهش سرعت نیز از طریق فرمول مسافت کل بصورت زیر بدست خواهد آمد.

$$D = d_a + d_c + d_d \rightarrow d_d = D - d_a - d_c$$

$$\left(\frac{1}{2} \times t_d \times V_{max}\right) = D - \left(\frac{1}{2} \times t_a \times V_{max}\right) - d_c$$

با توجه به اینکه زمان های t_a و t_d را قبلا محاسبه کرده بودیم، حال با جایگذاری آنها در معادله فوق فرمول زیر بدست می آید. اما قبل از آن باید توجه داشته باشید، زمان t_d که از فرمول شتاب محاسبه می شود برخلاف زمان t_a دارای مقدار منفی است چون در بخش کاهش سرعت شتاب منفی خواهد بود.

$$a = \frac{\Delta v}{t_d} = \frac{v_2 - v_1}{t_d} \rightarrow v_2 = 0 \ , \ v_1 = V_{max} \rightarrow a = -\frac{V_{max}}{t_d} \rightarrow t_d = -\frac{V_{max}}{a}$$

$$\left(\frac{1}{2} \times \left(-\frac{V_{max}}{a}\right) \times V_{max}\right) = D - \left(\frac{1}{2} \times \left(\frac{V_{max}}{a}\right) \times V_{max}\right) - d_c$$

حال با کمی ساده سازی معادله فوق، سرعت لحظه‌ای برای بخش کاهش سرعت بصورت زیر محاسبه خواهد شد.

$$\frac{(V_{max})^2}{2a} = \frac{(V_{max})^2}{2a} + d_c - D \;\rightarrow\; (V_{max})^2 = (V_{max})^2 + 2a(d_c - D)$$
$$V_{decel} = \sqrt{(V_{max})^2 + 2a(d_c - D)}$$

تا به الان تمام محاسبات پیاده سازی پروفایل حرکت ذوزنقه‌ای برای استپرموتور انجام شد حال نوبت به آن می رسد که این فرمول ها را در یکی از محیط های برنامه نویسی میکروکنترلر پیاده کنیم. نمونه برنامه ایجاد شده توسط فرمول های فوق به شکل زیر خواهد بود.

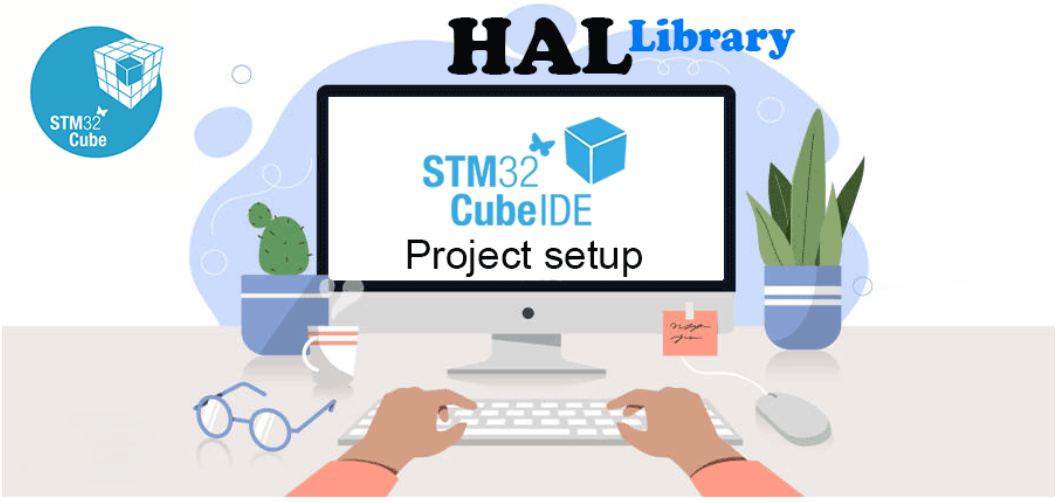
```
void Trapezoidal_Profile(float maxSpeed , float Acceleration , float Distance){

  // define end point of acceleration and start point of deceleration
  float da = maxSpeed * maxSpeed / (2 * Acceleration);
  float dc = Distance - da;

  if (da > dc) // if we don't even reach full speed
  {
    da = dc = Distance / 2;
    maxSpeed = sqrt(Distance * Acceleration);
  }

  float vcurr;
  for (uint32_t incStep = 0; incStep < Distance; incStep++)
  {
    // calculate central position of current step
    float Step = ((float) incStep + 0.5);

    // calculate velocity at current step
    if (Step < da) vcurr = sqrt(2 * Step * Acceleration);

    else if (Step < dc) vcurr = maxSpeed;

    else vcurr = sqrt(maxSpeed * maxSpeed + 2 * Acceleration * (dc - Step));

    // convert velocity to delay
    uint32_t Delay = round(1000000 / vcurr); // in microsecond

    StepperMotor_1_SetPulse;
    delay_us(3);
    StepperMotor_1_ResetPulse;
    delay_us(Delay);
  }
}
```

در فصل آینده که به معرفی و بررسی کدهای کنترلر ربات اسکارا خواهیم پرداخت، خواهید دید که در کتابخانه Motion Profile از این تابع برای کنترل حرکت ربات اسکارا استفاده می کنیم.

فصل ششم: برنامه نویسی کنترلر ربات اسکارا

۶-۱. پیکربندی میکروکنترلر STM32 در محیط Cube IDE

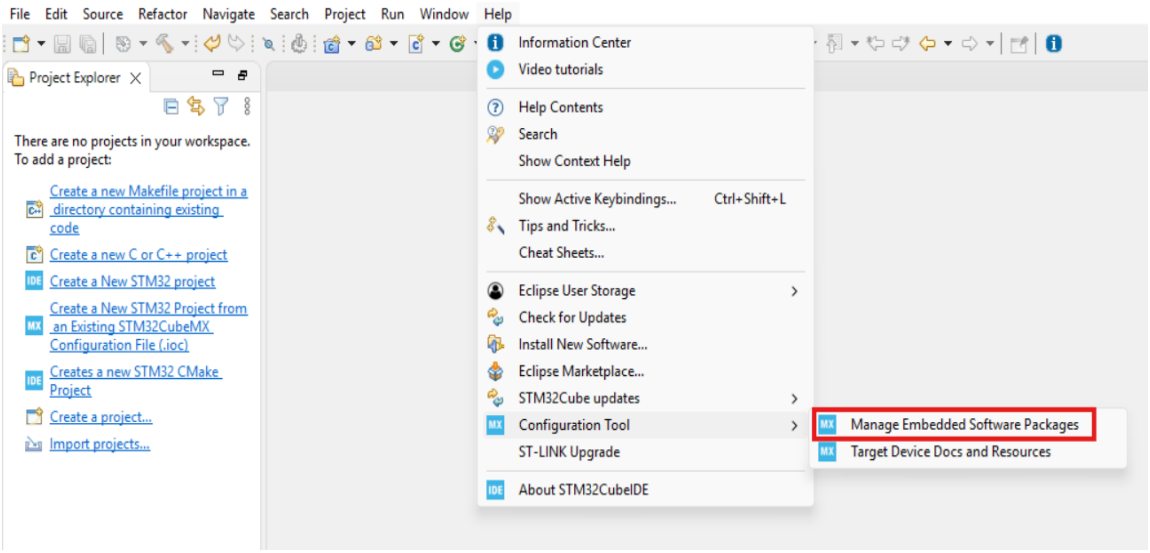
برای شروع برنامه نویسی میکروکنترلر STM32 ، ابتدا باید این میکروکنترلر پیکربندی شود و برای پیکربندی میکروکنترلر ما به نرم افزار Cube MX نیاز خواهیم داشت. البته همانطور که در فصل قبل در مورد آن بحث شد، شرکت ST ، نرم افزاری تحت عنوان Cube IDE بصورت رایگان در اختیار کاربران قرار داد که میتوان هم به عنوان یک محیط برنامه نویسی و هم به عنوان یک محیط گرافیکی برای پیکربندی میکروکنترلر استفاده نمود.



شکل ۶- ۱- پیکربندی میکروکنترلر STM32 در محیط Cube IDE

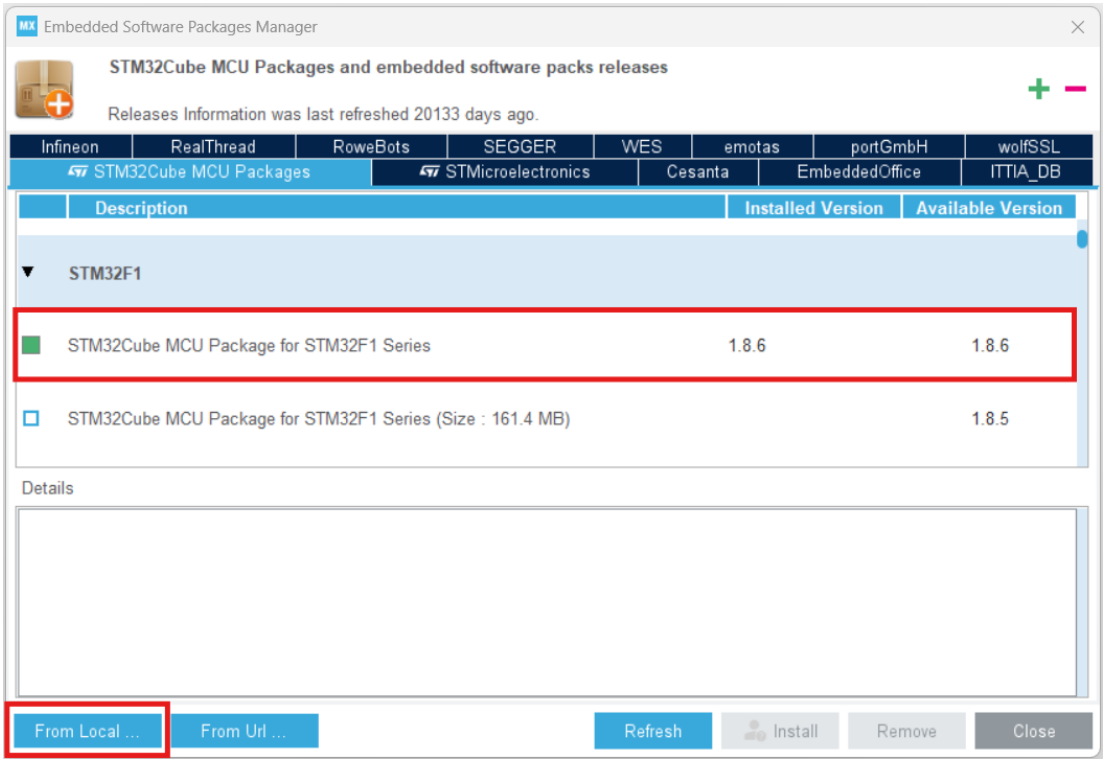
برای شروع پیکربندی میکروکنترلر STM32 ، ابتدا باید آخرین نسخه این نرم افزار را از سایت ST دانلود نمایید که البته بدلیل مشکلات فیلترینگ این سایت برای کاربران ایرانی، پیشنهاد می شود که از سایت های معتبر ایرانی همچون Soft98 و سیسوگ برای دانلود آخرین نسخه این نرم افزار استفاده نمایید. پس از نصب نرم‌افزار Cube IDE ، اولین کاری که باید انجام دهید این است که فریمورک یا همان پکیج نرم‌افزاری میکروکنترلری که می‌خواهید با آن کار کنید را دانلود نمایید. در اینجا چون ما از میکروکنترلر STM32F103C8T6 استفاده می‌کنیم، پس نیاز است که فریمورک مربوط به سری STM32F1 را دانلود نماییم. دقت کنید که برای هر سری از میکروکنترلرهای شرکت ST یک فریمورک وجود دارد که آن فریمورک قابل استفاده برای تمامی میکروکنترلرهای موجود در آن سری است. برای دانلود فریمورک می توانید از داخل خود نرم افزار نیز اقدام نمایید اما مجددا به دلیل مشکل فیلترنیگ، پیشنهاد می شود از سایت سیسوگ آخرین نسخه پکیج نرم افزاری میکروکنترلر مربوطه را دانلود نمایید. برای نصب این پکیج نیز می توانید طبق مراحل زیر اینکار را انجام دهید.

ابتدا مانند شکل ۶-۲ از منوی Help در بخش Configuration Tool گزینه Manege embedded software packages را انتخاب کنید.



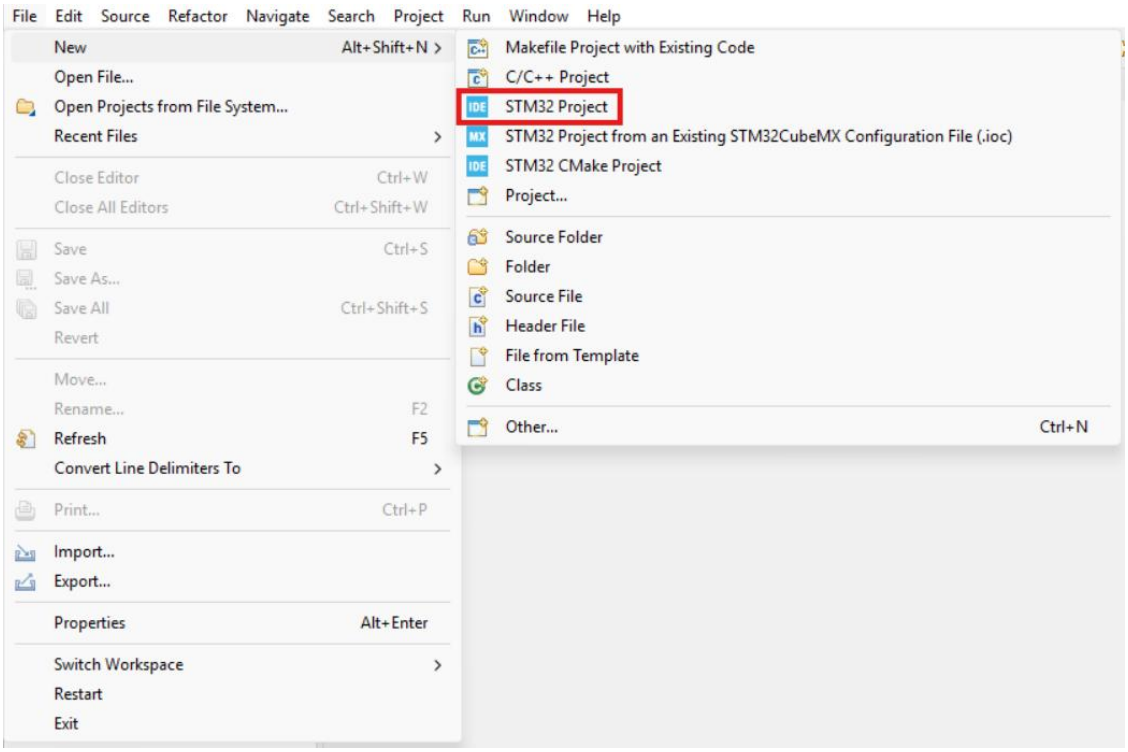
شکل ۶- ۲- بخش مدیریت پکیج های نرم افزاری

سپس در پنجره باز شده، سری STM32F1 را باز نموده و در پایین صفحه گذینه From Local را انتخاب کنید تا بتوانید فایل پکیج نرم افزاری که قبلا دانلود کرده اید را نصب نمایید. پس از اینکه نصب پکیج بطور کامل انجام گردید، مربع کنار پکیج مربوطه به رنگ سبز تغییر خواهد کرد.



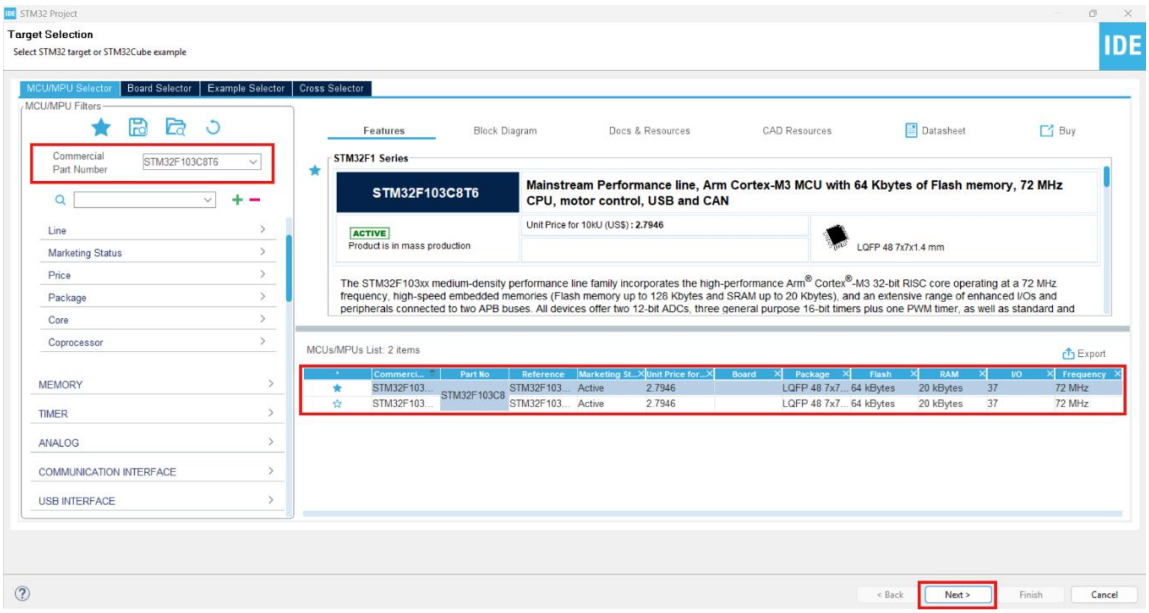
شکل ۶- ۳- نصب پکیج نرم افزاری بصورت دستی

پس از اتمام نصب وقت آن است که یک پروژه ایجاد کنیم. برای ایجاد پروژه مانند شکل ۶-۴ از منوی File در سربرگ New گزینه STM32 Project را انتخاب نمایید.



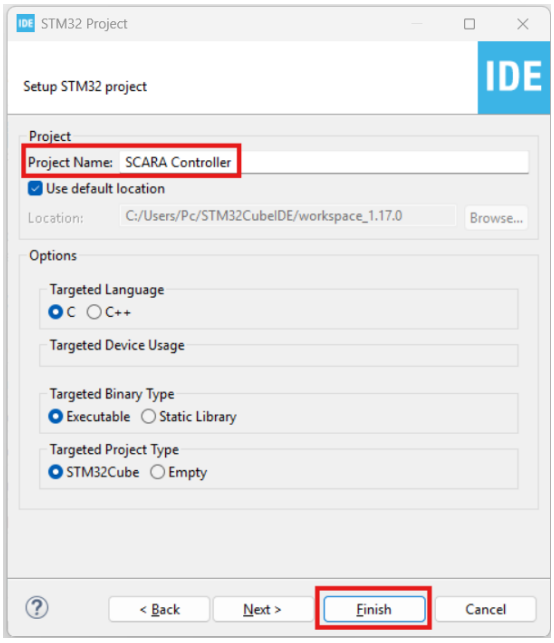
شکل ۶- ۴- نحوه ایجاد پروژه جدید

در پنجره باز شده، ابتدا در قسمت Commercial Part Number مدل میکروکنترلر خود را جستجو و سپس در صفحه کناری، آنرا انتخاب کنید و درنهایت در پایین صفحه گذینه Next را انتخاب نمایید تا به بخش بعدی هدایت شوید.

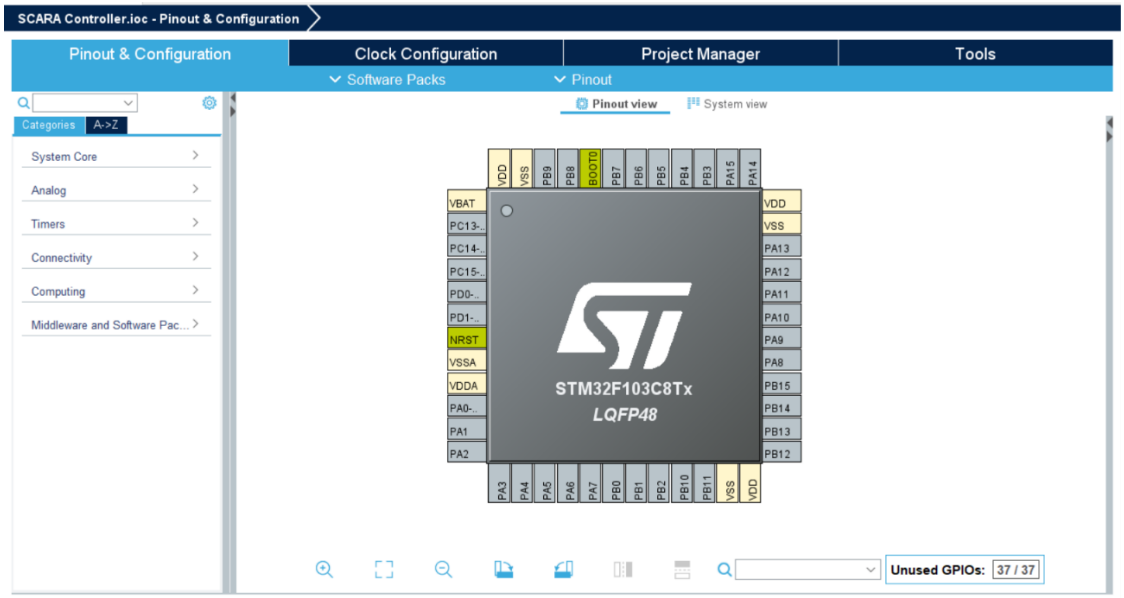


شکل ۶- ۵- انتخاب نوع و مدل میکروکنترلر

در بخش بعدی، نیاز به تغییرات خاصی نیست و فقط با انتخاب نام پروژه و انتخاب گذینه Finish پروژه شما ساخته خواهد شد و وارد محیط پیکربندی میکروکنترلر STM32 می شود.



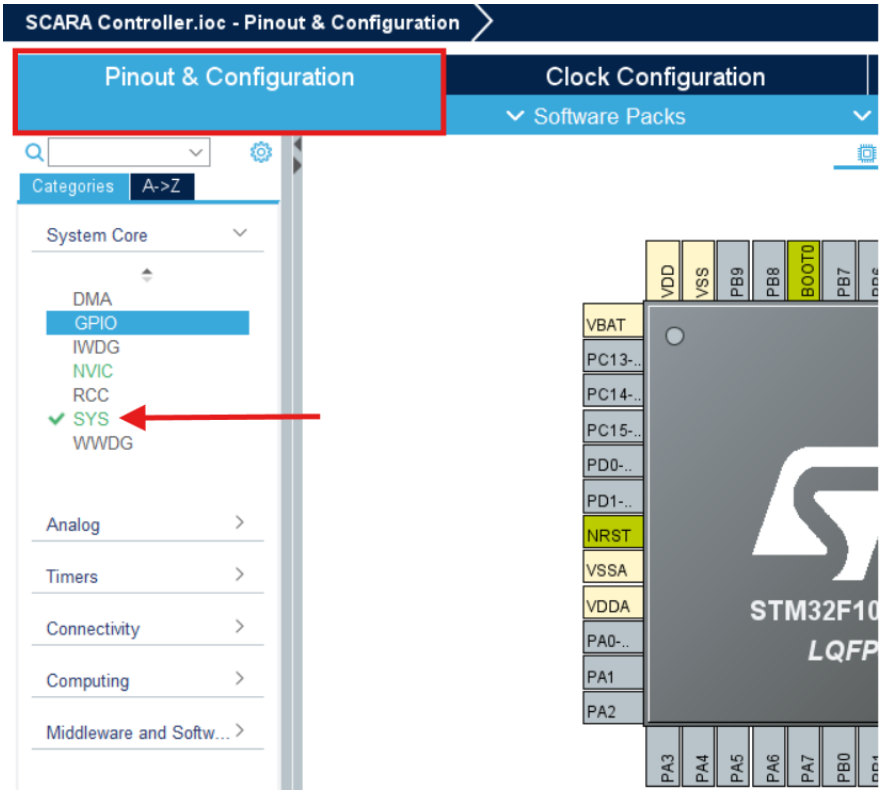
شکل ۶- ۶- تعیین نام برای ایجاد پروژه



شکل ۶- ۷- محیط پیکربندی میکروکنترلر

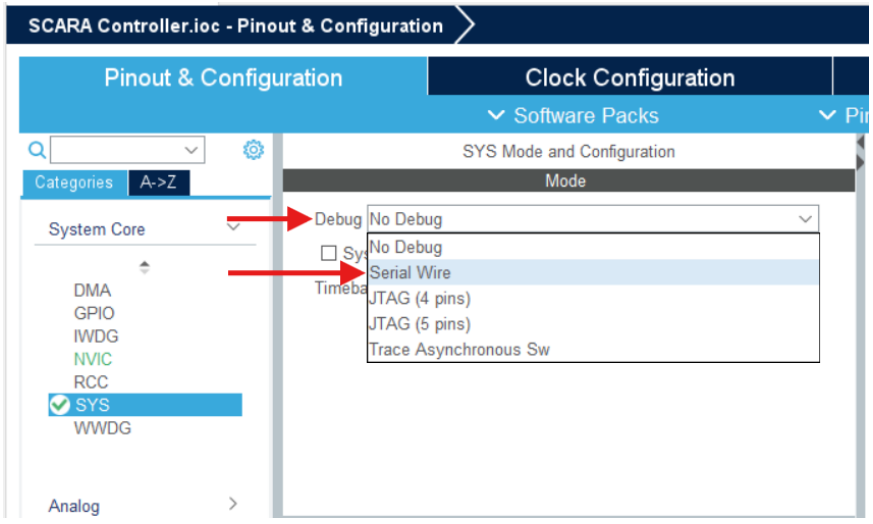
همانطور که در شکل ۶-۷ مشاهده می کنید این محیط شامل بخش های مختلفی است که در ادامه به توضیح هر کدام از بخش های مهم و مورد نیاز برای ساخت پروژه کنترلر ربات اسکارا پرداخته خواهد شد، البته پیشنهاد می شود که ابتدا از منابع تصویری و ویدیویی موجود در اینترنت برای آموزش کامل این محیط استفاده نمایید سپس در ادامه با ما همراه باشید.

برای شروع ابتدا در سربرگ Pinout & Configuration وارد بخش SYS شوید، این بخش مربوط به تنظیم نوع یا روش پروگرام کردن میکروکنترلر می باشد.



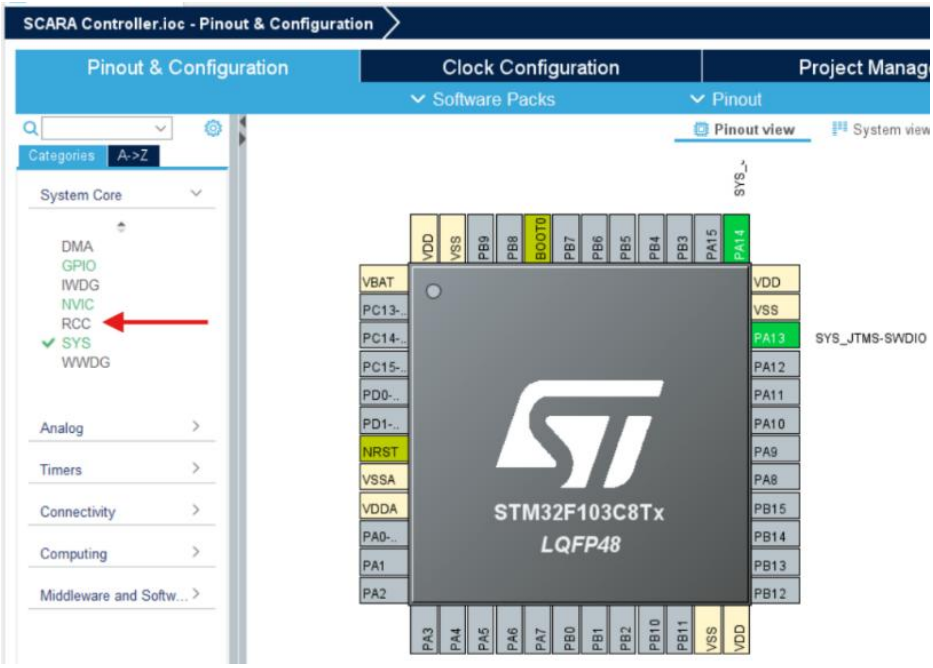
شکل ۶- ۸- انتخاب روش پروگرام کردن میکروکنترلر

معمولا میکروکنترلرهای ST را به دو روش JTAG و SWD پروگرام می‌کنند. در روش SWD نیاز به پروگرامر ST-Link و در روش JTAG نیاز به پروگرامر J-Link داریم، توضیحات بیشتر در فصل قبل داده شده است. همانطور که در شکل ۶-۹ مشاهده می‌کنید، باید گزینه Debug را بر روی Serial Wire تنظیم کنید تا بتوانیم از روش SWD برای پروگرام کردن میکروکنترلر استفاده نماییم.



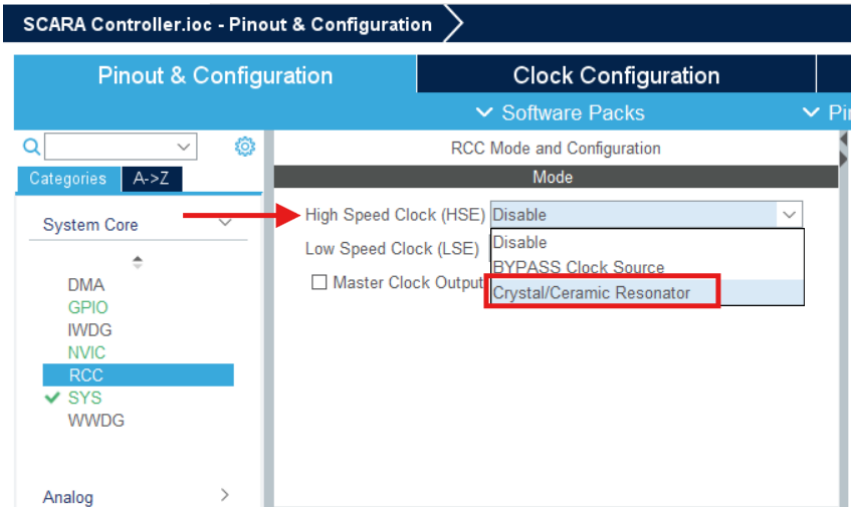
شکل ۶- ۹- تنظیم روش SWD یا Serial Wire

پس از تنظیم نوع پروگرامر به سراغ تنظیمات کلاک میکروکنترلر رفته و در بخش System Core گذینه RCC یا همان Reset and Clock Control را انتخاب می‌کنیم.



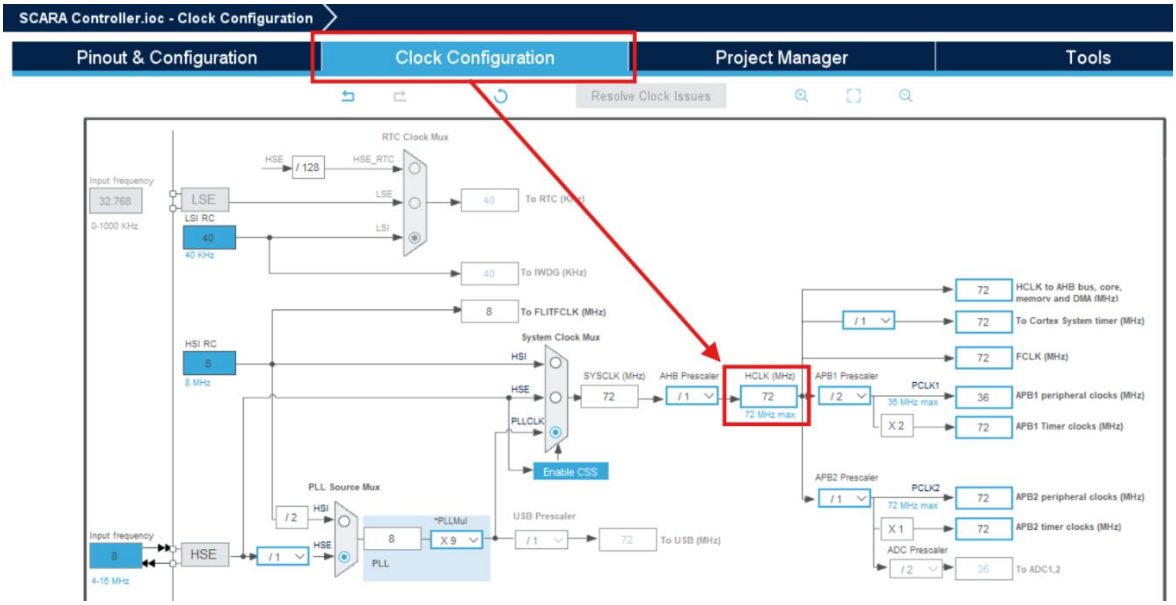
شکل ۶- ۱۰- تنظیمات کلاک میکروکنترلر

در صورتی که بخواهیم از کریستال خارجی استفاده کنیم تنظیمات زیر را انجام می‌دهیم. البته توجه داشته باشید که در حالت عادی تنظیمات کلاک میکروکنترلر روی گزینه HSI یا همان کلاک سریع داخلی با فرکانس ۸ مگاهرتز می باشد و حداکثر می تواند تا فرکانس ۶۴ مگاهرتز تنظیم شود، به همین دلیل برای افزایش آن به فرکانس ۷۲ مگاهرتز باید از کریستال خارجی ۸ مگاهرتز که بر روی برد توسعه STM32 تعبیه شده است استفاده نمایید. بدین منظور بعد از وارد شدن به بخش RCC ، گزینه High Speed Clock (HSE) را بر روی Crystal/Ceramic Resonator تنظیم نمایید.



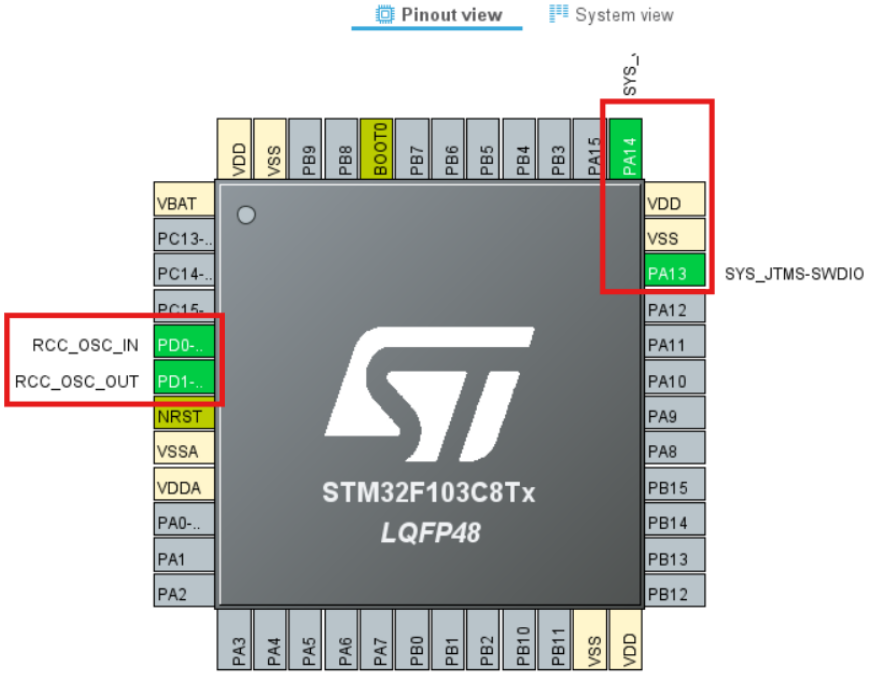
شکل ۶- ۱۱- انتخاب کریستال خارجی به عنوان کلاک میکروکنترلر

پس از اینکه تنظیمات فوق را انجام دادید وارد بخش Clock Configuration شوید و در کادر HCLK حداکثر فرکانس میکروکنترلر را بر روی ۷۲ مگاهرتز قرار دهید.



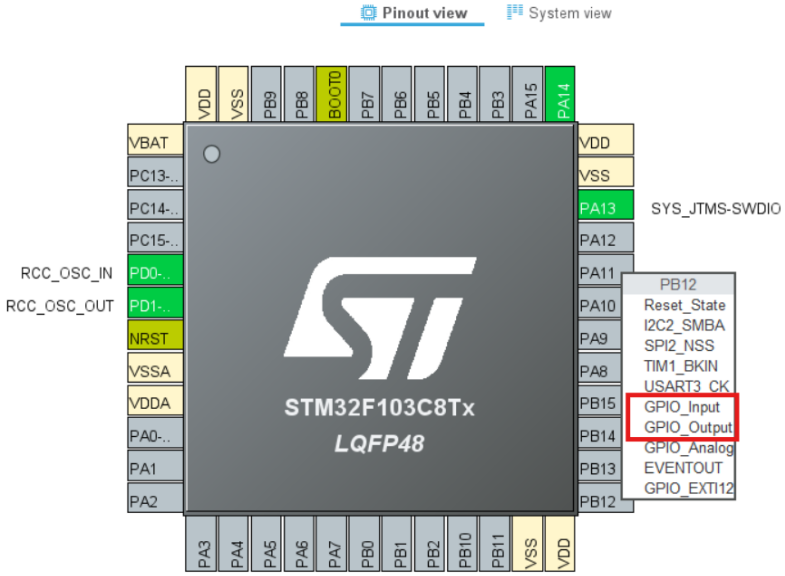
شکل ۶- ۱۲- تنظیم حداکثر فرکانس میکروکنترلر

درنهایت می توانید نتیجه تنظیمات انجام شده را در بخش Pinout view بر روی میکروکنترلر خود مشاهده نمایید. همانطور که در شکل ۶-۱۳ مشخص است، پایه های PA13 و PA14 برای اتصال پروگرامر و پایه های PD0 و PD1 برای اتصال کریستال خارجی در نظر گرفته شده اند.



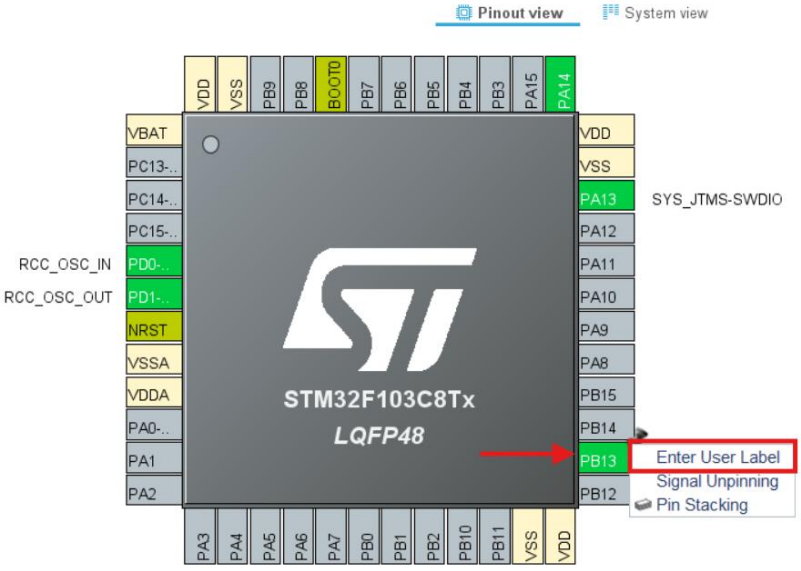
شکل ۶- ۱۳- پایه های میکروکنترلر پس از اعمال تنظیمات

با توجه به مدل میکروکنترلری که انتخاب می‌کنید، تعداد مشخصی GPIO در اختیار شما است که می توانید این GPIO ها را به صورت ورودی و یا خروجی تعریف کنید. برای تعریف GPIO بصورت ورودی و یا خروجی بایستی بر روی یکی از پایه های میکروکنترلر کلیک کنید تا گزینه های قابل تنظیم GPIO را مشاهده نمایید. سپس با کلیک بر روی GPIO_Output پایه مورد نظر خروجی و با انتخاب GPIO_Input پایه بصورت ورودی تعریف می‌شود.



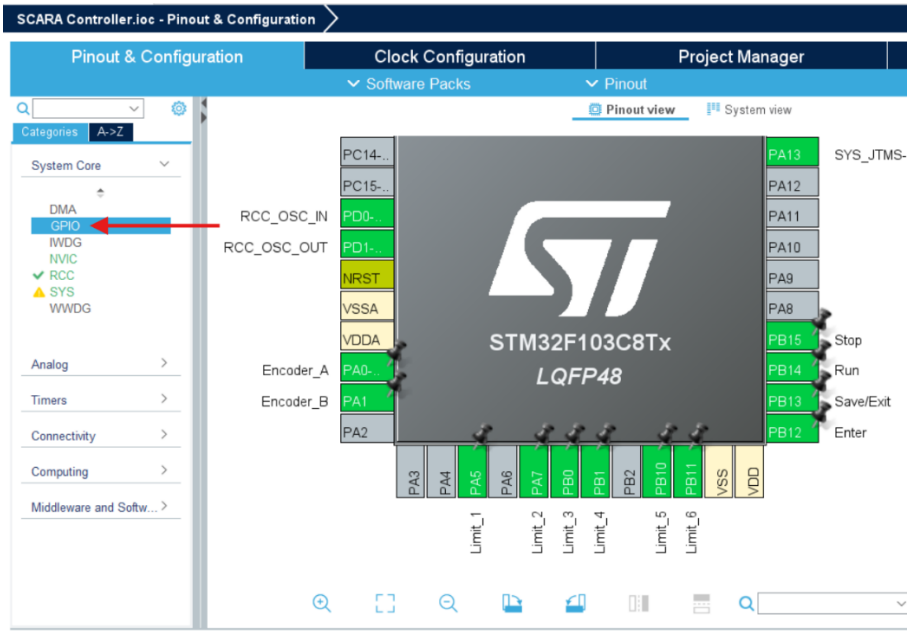
شکل ۶- ۱۴- تنظیم پایه مورد نظر به عنوان ورودی یا خروجی

بهتر است برای هر پایه ای که استفاده می‌کنید یک نام یا لیبل انتخاب کنید تا در بخش کدنویسی راحت‌تر بتوان از پایه ها استفاده نمود. برای انتخاب نام بایستی بر روی پایه ای که استفاده نموده اید راست کلیک و گزینه Enter User Label را انتخاب کنید و سپس نام مورد نظر را وارد نمایید و Enter بزنید.



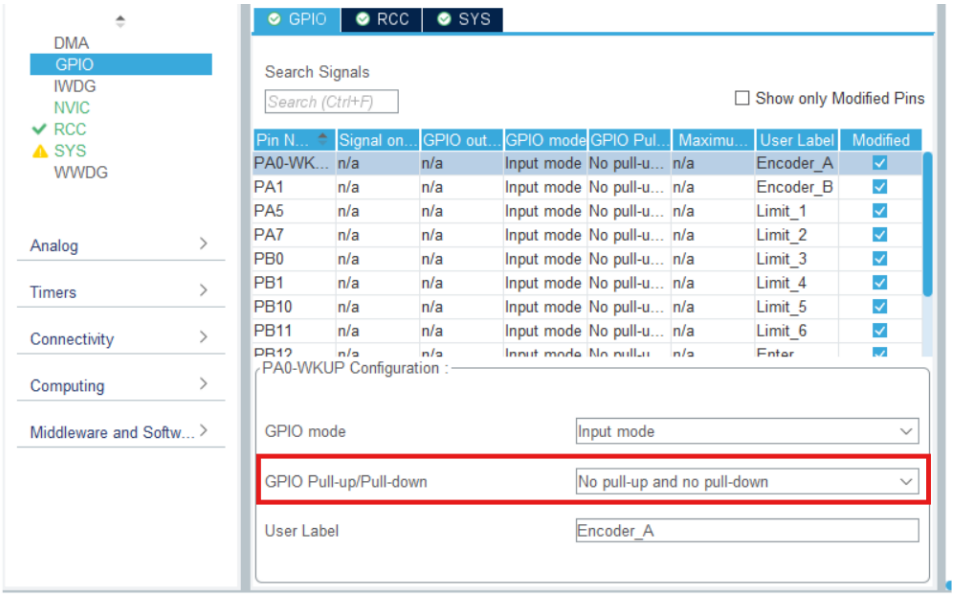
شکل ۶- ۱۵- انتخاب نام حذبرای پایه انتخاب شده

در این پروژه ما از پایه های PB12 تا PB15 را برای کلیدهای کنترلی و پایه های PA0 و PA1 را برای روتاری انکودر و پایه های PA5 و PA7 و PB0 و PB1 و PB10 و PB11 را برای سنسورها به عنوان ورودی میکروکنترلر در نظر می گیریم.



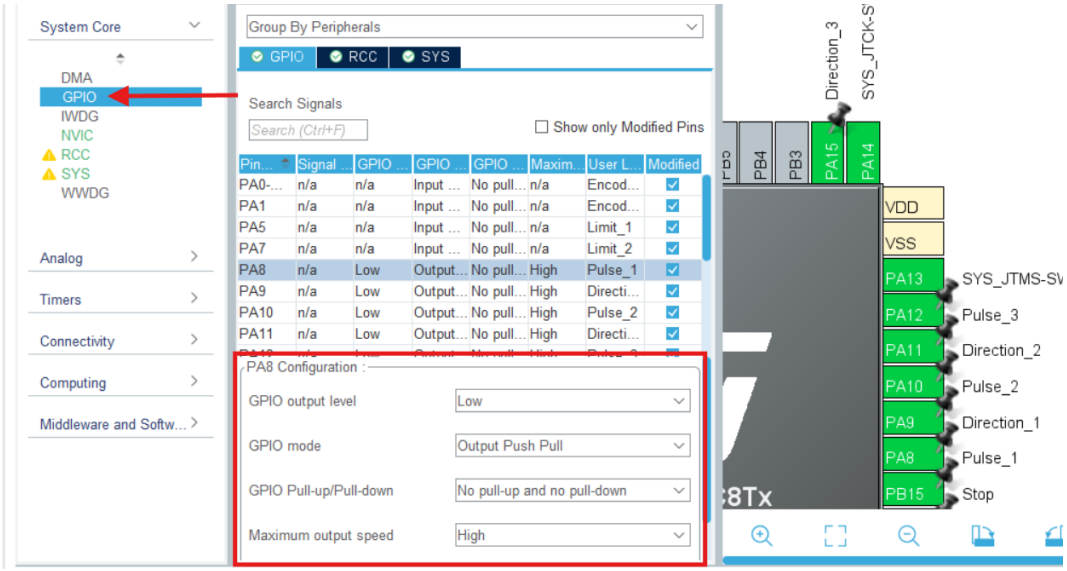
شکل ۶- ۱۶- تعیین پایه های ورودی میکروکنترلر

پس از تنظیم پایه های میکروکنترلر به عنوان ورودی، می توانید در بخش GPIO تنظیمات تکمیلی را نیز انجام دهید. بدین منظور تنها پارامتری که میتوان در این بخش تغییر داد مربوط به تعیین حالت ورودی به صورت Pull-up یا Pull-down می باشد و چون در کنترلر این پایه ها بصورت فیزیکی Pull-up شده اند نیازی به تغییر این تنظیمات در این بخش نخواهیم داشت.



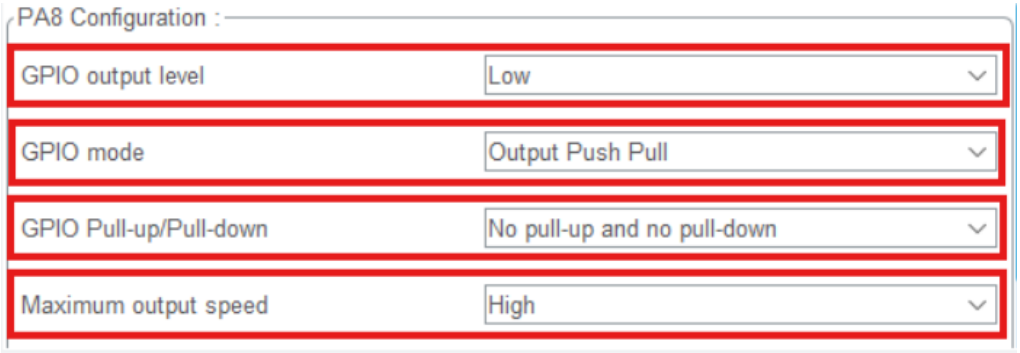
شکل ۶- ۱۷- تنظیم نوع ورودی میکروکنترلر

در این پروژه ما از پایه های PA8 تا PA15 را برای ارسال پالس به درایو استپرموتور به عنوان خروجی میکروکنترلر در نظر می گیریم. بدین صورت ما به سه گروه خروجی Pulse و Direction برای کنترل سه استپر موتور یا سه محور ربات اسکارا نیاز خواهیم داشت.



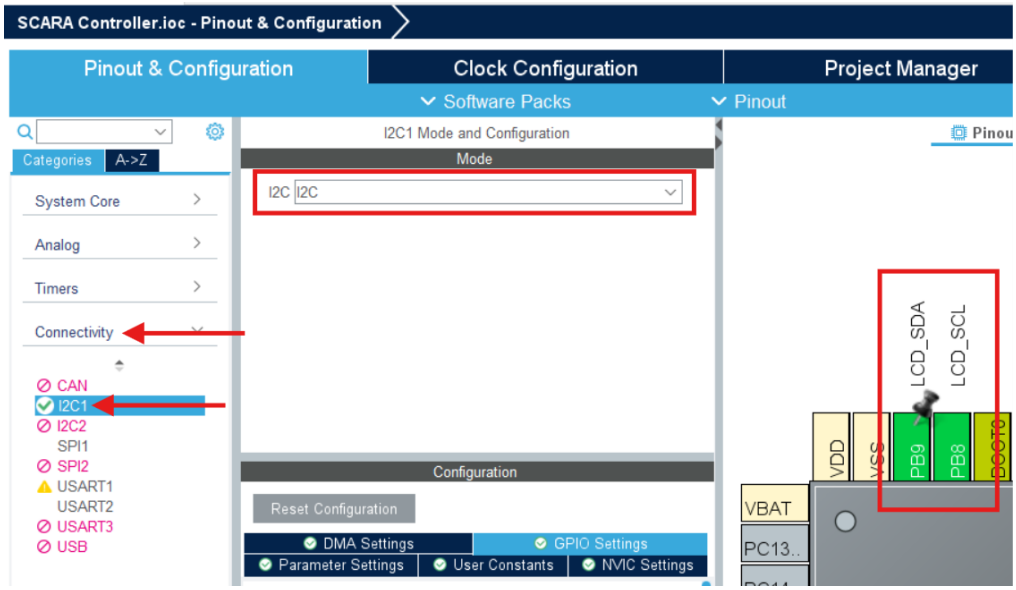
شکل ۶- ۱۸- تعیین پایه های خروجی میکروکنترلر

پس از تنظیم پایه های میکروکنترلر به عنوان خروجی، می توانید در بخش GPIO تنظیمات تکمیلی را نیز انجام دهید. همانطور که در شکل ۶-۱۹ مشاهده می کنید، این تنظیمات شامل چهار بخش می باشد که اولین پارامتر آن مربوط به تنظیم سطح سیگنال پیشفرض خروجی می باشد که باید بر روی Low قرار داشته باشد و دومین پارامتر مربوط به تعیین حالت خروجی بصورت Push Pull یا Open Drain می باشد که باید بر روی Push Pull قرار داشته باشد و سومین پارامتر نیز همانند تنظیمات ورودی، نیازی به تغییر نیست و چهارمین پارامتر مربوط به تعیین سرعت سوئیچینگ می باشد که باید بر روی High قرار داشته باشد.



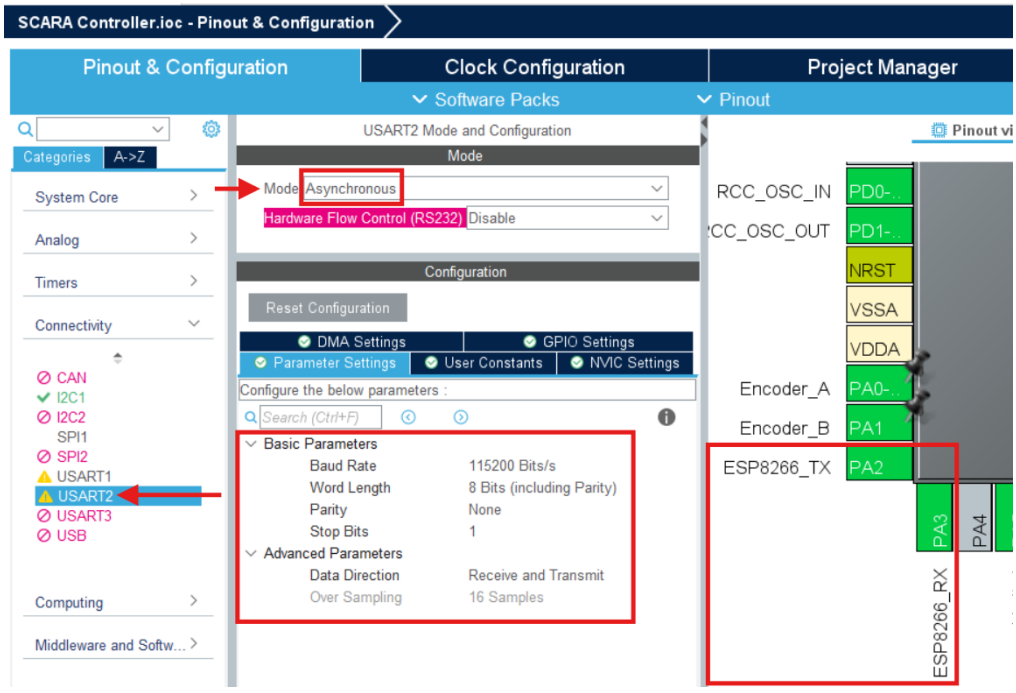
شکل ۶- ۱۹- تنظیمات پایه خروجی میکروکنترلر

برای اتصال LCD کاراکتری به میکروکنترلر نیاز به فعالسازی واحد I2C خواهید داشت، بدین صورت برای فعالسازی این واحد باید از سربرگ Connectivity در بخش I2C1 اقدام کنید. پس از فعالسازی می توانید تغییرات اعمال شده بر روی پایه های میکروکنترلر را مشاهده نمایید.



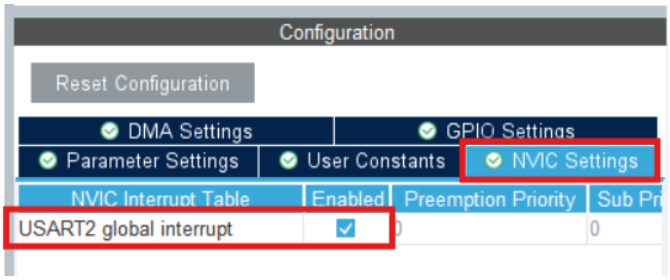
شکل ۶- ۲۰- فعالسازی واحد I2C میکروکنترلر

برای اتصال ماژول ESP8266 به میکروکنترلر نیاز به فعالسازی واحد USART خواهید داشت، بدین صورت برای فعالسازی این واحد باید از سربرگ Connectivity در بخش USART2 اقدام کنید. پس از وارد شدن به این بخش باید گذینه Mode را بر روی Asynchronous یا آسنکرون قرار دهید.



شکل ۶- ۲۱- فعالسازی واحد USART میکروکنترلر

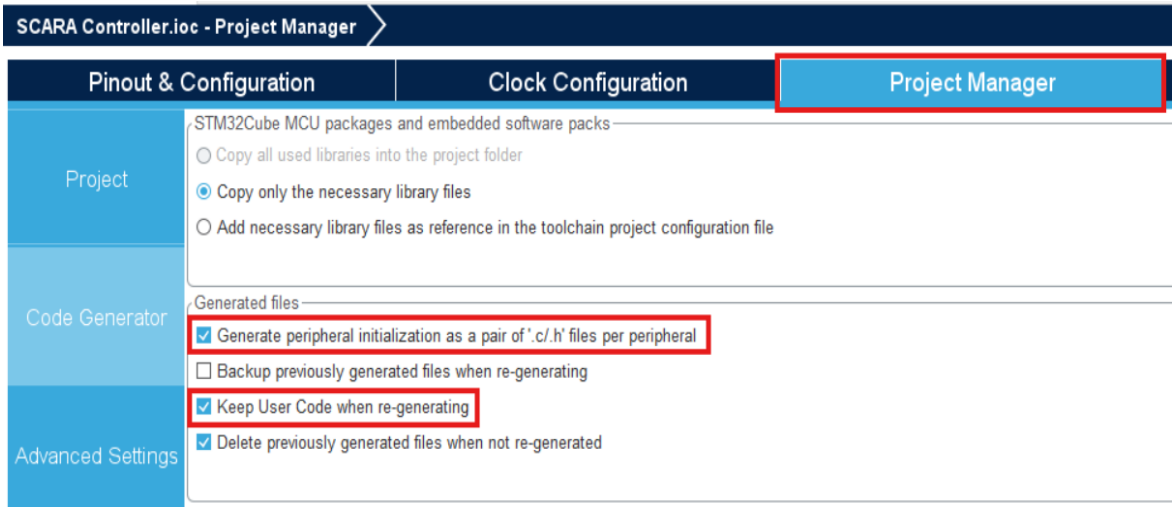
همانطور که در شکل ۶-۲۱ مشاهده می کنید در سربرگ Parameter Settings یکسری پارامترهای پایه برای ارتباط سریال وجود دارد که لازم است آنها را متناسب با نیاز خود تنظیم نمایید. درنهایت حتما گذینه USART2 global interrupt در سربرگ NVIC Settings را نیز انتخاب کنید تا واحد وقفه برای ارتباط سریال فعالسازی شود.



شکل ۶- ۲۲- فعالسازی وقفه سراسری برای واحد USART

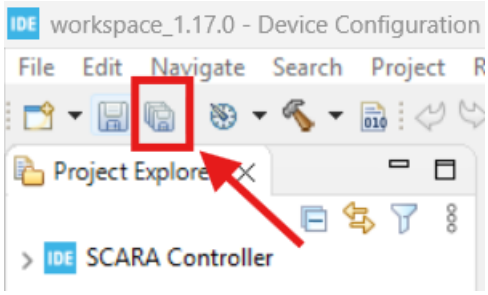
به این نکته نیز حتما توجه کنید که علت انتخاب واحد USART2 به این دلیل است که سطح ولتاژ منطقی آن ۳.۳ ولت است و مابقی واحد های USART با ولتاژ ۵ ولت کار می کنند و چون ماژول ESP8266 با ولتاژ ۳.۳ ولت کار می کند نیاز هست که از واحد USART2 استفاده نماییم.

آخرین مرحله ای که نیاز به اعمال تغییرات در آن هستیم مربوط به بخش Project Manager می باشد، بدین منظور وارد بخش Code Generator از سربرگ Project Manager شده و گذینه Generate peripheral initialization انتخاب کنید تا نرم افزار Cube IDE برای هر واحد فعالسازی شده در محیط پیکربندی مانند واحد GPIO و I2C و USART یک کتابخانه مجزا در نظر بگیرد تا دسترسی سریع تر و سازماندهی شده به کدهای مربوط به این واحدها داشته باشیم. در قدم بعدی گذینه Keep User Code when re-generating را نیز انتخاب نمایید تا با هر بار تنظیم و پیکره‌بندی دوباره در این محیط، کدهایی که در محیط برنامه نویسی توسط کاربر نوشته می‌شوند حذف نشوند.



شکل ۶- ۲۳- تنظیمات بخش Project Manager

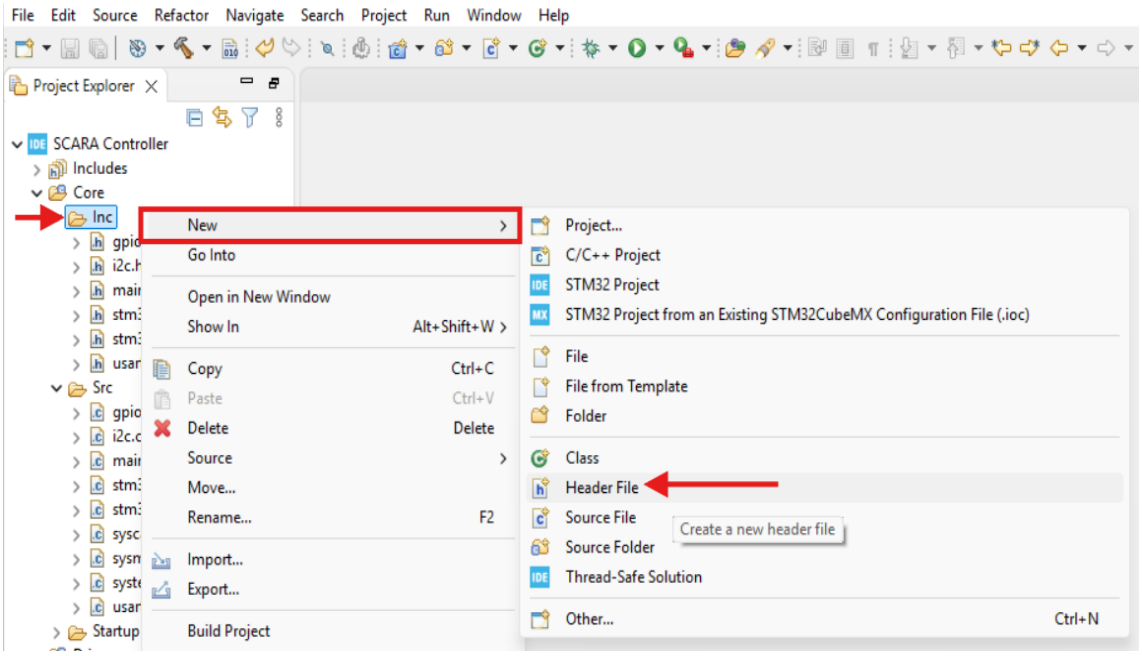
با انجام تنظیمات فوق، تمام تنظیمات مربوط به پیکربندی میکروکنترلر STM32 را در نرم افزار Cube IDE را انجام دادیم، درنهایت باید این پروژه را ذخیره کنیم تا تغییرات بخش پیکربندی بر روی کل پروژه اعمال شود. برای این منظور از طریق نوار ابزار بالای صفحه گذینه Save All را انتخاب نمایید.



شکل ۶- ۲۴- ذخیره تمام تغییرات اعمال شده

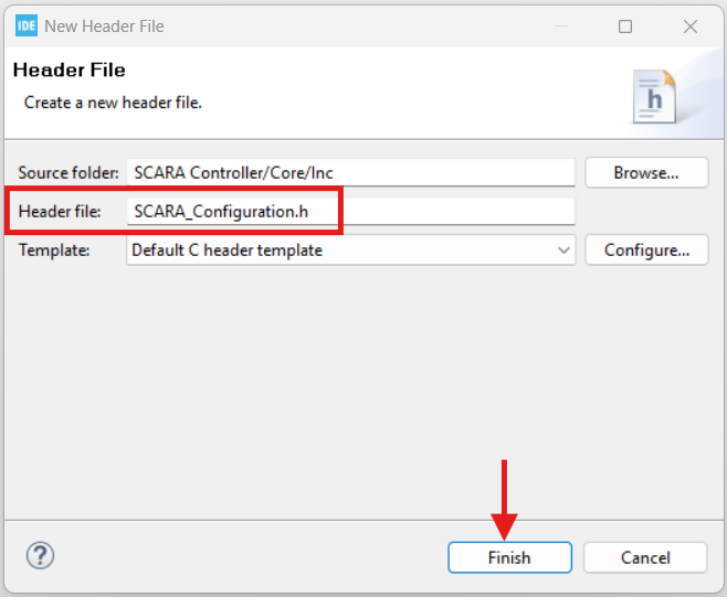
برای شروع برنامه نویسی اولین کاری که لازم است انجام دهیم ایجاد کتابخانه برای هر بخش برنامه می باشد، اینکار علاوه بر سازماندهی کدها باعث می شود حجم برنامه نیز کاهش پیدا کند. برای ایجاد کتابخانه ما به دو فایل با پسوند h. و c. نیاز خواهیم داشت، به فایل هایی که با پسوند h. ساخته می شوند اصطلاحاً Header file و به فایل هایی که با پسوند c. ساخته می شوند اصطلاحاً Source file می گویند. بدین معنا که در هدر فایل، معمولا پیکربندی و مقداردهی اولیه نظیر فراخوانی کتابخانه های دیگر، ایجاد ماکروها و نمونه اولیه توابع قرار می گیرند و در سورس فایل نیز کد اصلی شامل متغیرها و توابع قرار دارند.

برای ایجاد کتابخانه در محیط Cube IDE باید از منوی Project Explorer بر روی پوشه Inc کلیک راست کنید و در بخش New ، گذینه Header File را انتخاب کنید.



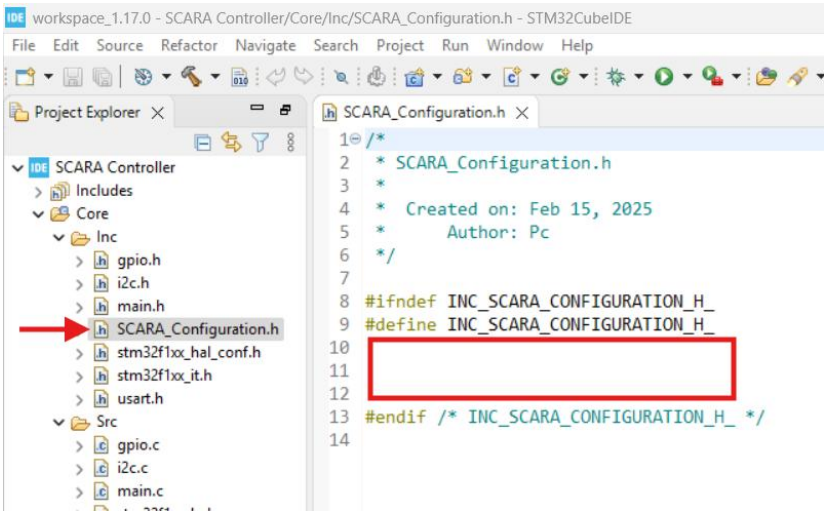
شکل ۶- ۲۵- نحوه ایجاد هدر فایل

سپس در پنجره باز شده باید یک نام برای هدر فایل خود انتخاب نمایید. دقت کنید حتما در انتهای نام انتخابی پسوند h. را نیز قرار دهید، پس از آن می توانید با کلیک بر روی گذینه Finish فایل خود را ایجاد کنید.



شکل ۶- ۲۶- انتخاب نام برای ایجاد هدر فایل

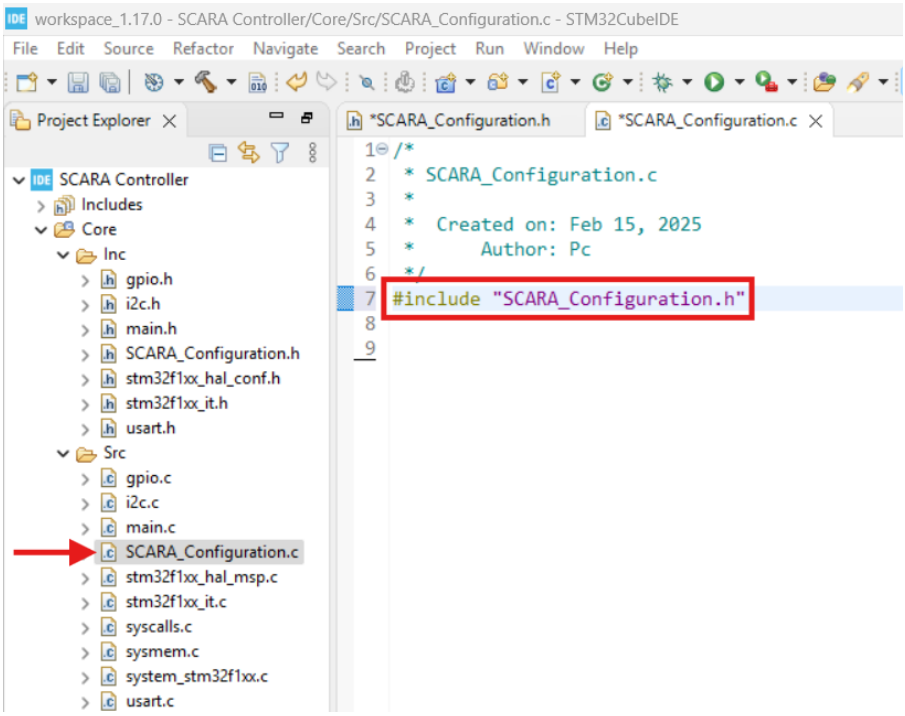
پس از ایجاد هدر فایل، می توانید فایل خود را در پوشه Inc مشاهده نمایید و در کادر علامت زده شده در شکل ۶-۲۷ نیز می توانید کدهای خود بنویسید.



شکل ۶- ۲۷- محیط هدر فایل

برای ایجاد سورس فایل نیز می توانید طبق مراحل قبل از منوی Project Explorer ، این بار بر روی پوشه Src کلیک راست کرده و در بخش New ، گذینه Source File را انتخاب کنید. همچنین در پنجره باز شده نیز باید یک نام برای سورس فایل خود با پسوند c. انتخاب نمایید.

پس از ایجاد سورس فایل، می توانید فایل خود را در پوشه Src مشاهده نمایید و حتما قبل از اینکه کدهای خود را در این محیط بنویسید، طبق شکل ۶-۲۸ هدر فایل خود را با پیشوند #include فراخوانی کنید.



شکل ۶- ۲۸- محیط سورس فایل

۶-۲. معرفی و بررسی کتابخانه SCARA Configuration

همانطورکه قبلا گفتیم برای شروع برنامه نویسی نیاز به ایجاد کتابخانه برای سازماندهی برنامه داریم و برای اینکار ما کتابخانه SCARA Configuration را ایجاد کردیم. این کتابخانه یک کتابخانه جامع برای جایگذاری کدهای اصلی و پیکربندی بخش های مهم برنامه می باشد. برای شروع کار ابتدا نیاز است که یکسری کتابخانه داخلی را فراخوانی کنیم. این کتابخانه ها بصورت پیشفرض در پروژه قرار دارند و نیازی به ایجاد آنها نیست.

```
/*----------------------[INTERNAL LIBRARIES]----------------------*/
#include "stdio.h"
#include "stdbool.h"
#include "stdlib.h"
#include "string.h"
#include "math.h"
```

سپس کتابخانه های خارجی که توسط خودمون ساخته شده است را نیز فراخوانی می کنیم. این کتابخانه ها حتما باید به پروژه اضافه شوند، در ادامه به معرفی و بررسی هر کتابخانه خواهیم پرداخت.

```
/*----------------------[EXTERNAL LIBRARIES]----------------------*/
#include "micros.h"
#include "I2C_LCD.h"
#include "Basic_Setting.h"
#include "Advanced_Setting.h"
#include "Program_Setting.h"
#include "Homing_Setting.h"
#include "Motion_Control.h"
#include "Calculator.h"
#include "Server.h"
```

پس از فراخوانی کتابخانه ها نیاز داریم که پایه های ورودی میکروکنترلر را در برنامه تعریف نماییم، برای اینکار می توانیم از ماکروها یا دستورات پیش پردازنده استفاده کنیم تا کدها تمیز و خوانا باشند. برای اینکه مقدار ورودی را از میکروکنترلر بخوانیم باید از تابع HAL_GPIO_ReadPin استفاده کنیم، بدین صورت پایه های ورودی میکروکنترلر را به شکل زیر در برنامه وارد می کنیم.

```
/*----------------------[INTERNAL DEFINITIONS]----------------------*/
#define Run_Button  HAL_GPIO_ReadPin(GPIOB , Run_Pin)
#define Stop_Button  HAL_GPIO_ReadPin(GPIOB , Stop_Pin)
#define Enter_Button  HAL_GPIO_ReadPin(GPIOB , Enter_Pin)
#define SaveExit_Button  HAL_GPIO_ReadPin(GPIOB , Save_Exit_Pin)

#define Encoder_A  HAL_GPIO_ReadPin(GPIOA , Encoder_A_Pin)
#define Encoder_B  HAL_GPIO_ReadPin(GPIOA , Encoder_B_Pin)

#define LimitJoint_1A  HAL_GPIO_ReadPin(GPIOA , Limit_1_Pin)
#define LimitJoint_1B  HAL_GPIO_ReadPin(GPIOA , Limit_2_Pin)
#define LimitJoint_2A  HAL_GPIO_ReadPin(GPIOB , Limit_3_Pin)
#define LimitJoint_2B  HAL_GPIO_ReadPin(GPIOB , Limit_4_Pin)
#define LimitZAxis_1A  HAL_GPIO_ReadPin(GPIOB , Limit_5_Pin)
#define LimitZAxis_1B  HAL_GPIO_ReadPin(GPIOB , Limit_6_Pin)
```

برای اینکه بتوانیم وضعیت پایه های خروجی میکروکنترلر را تغییر دهیم این بار هم باید از توابع هال استفاده نماییم، این تابع HAL_GPIO_WritePin نام دارد که در دو حالت SET و RESET استفاده می شود، بنابراین پایه های خروجی میکروکنترلر بصورت زیر در برنامه تعریف خواهند شد.

```
#define StepperMotor_1_SetPulse  HAL_GPIO_WritePin(GPIOA, Pulse_1_Pin, GPIO_PIN_SET)
#define StepperMotor_1_ResetPulse  HAL_GPIO_WritePin(GPIOA, Pulse_1_Pin, GPIO_PIN_RESET)
#define StepperMotor_1_SetDirection  HAL_GPIO_WritePin(GPIOA, Direction_1_Pin, GPIO_PIN_SET)
#define StepperMotor_1_ResetDirection  HAL_GPIO_WritePin(GPIOA, Direction_1_Pin, GPIO_PIN_RESET)

#define StepperMotor_2_SetPulse  HAL_GPIO_WritePin(GPIOA, Pulse_2_Pin, GPIO_PIN_SET)
#define StepperMotor_2_ResetPulse  HAL_GPIO_WritePin(GPIOA, Pulse_2_Pin, GPIO_PIN_RESET)
#define StepperMotor_2_SetDirection  HAL_GPIO_WritePin(GPIOA, Direction_2_Pin, GPIO_PIN_SET)
#define StepperMotor_2_ResetDirection  HAL_GPIO_WritePin(GPIOA, Direction_2_Pin, GPIO_PIN_RESET)

#define StepperMotor_3_SetPulse  HAL_GPIO_WritePin(GPIOA, Pulse_3_Pin, GPIO_PIN_SET)
#define StepperMotor_3_ResetPulse  HAL_GPIO_WritePin(GPIOA, Pulse_3_Pin, GPIO_PIN_RESET)
#define StepperMotor_3_SetDirection  HAL_GPIO_WritePin(GPIOA, Direction_3_Pin, GPIO_PIN_SET)
#define StepperMotor_3_ResetDirection  HAL_GPIO_WritePin(GPIOA, Direction_3_Pin, GPIO_PIN_RESET)

#define StepperMotor_1_TogglePulse  HAL_GPIO_TogglePin(GPIOA, Pulse_1_Pin)
#define StepperMotor_2_TogglePulse  HAL_GPIO_TogglePin(GPIOA, Pulse_2_Pin)
#define StepperMotor_3_TogglePulse  HAL_GPIO_TogglePin(GPIOA, Pulse_3_Pin)
```

درنهایت توابعی که در سورس فایل نیز برای نوشتن کدهای اصلی برنامه به آنها نیاز خواهیم داشت را در این بخش تعریف می نماییم.

```
//-----[ Prototypes For User External Functions ]-----
void ControllerSetup(void);
void ControllerSetting(void);
void ReadEncoder(void);
```

۶-۲-۱. معرفی متغیرهای خارجی

متغیرهای خارجی، متغیرهایی هستند که در کتابخانه های دیگر تعریف و مقداردهی اولیه شدند و در اینجا برای خواندن و تغییر مقدار آنها استفاده می شوند. این متغیرها بوسیله دستور extern به این کتابخانه فراخوانی می شوند.

```
/*----------------------[EXTERNAL VARIABLES]----------------------*/
extern UART_HandleTypeDef huart2;

extern uint8_t rxByte;

extern uint16_t Pulse_rev[3];

extern float speedValue[3];
extern float xPosition[2] , yPosition[2];
```

۶-۲-۲. معرفی متغیرهای داخلی

متغیرهای داخلی، متغیرهایی هستند که در این کتابخانه تعریف و مقداردهی اولیه شدند. این متغیرها می توانند در این کتابخانه یا کتابخانه های دیگر نیز استفاده شوند.

```
/*----------------------[INTERNAL VARIABLES]----------------------*/
const uint8_t degree[] = {0x07,0x05,0x07,0x00,0x00,0x00,0x00,0x00};
char *settingMenu [4] = {"Basic Setting" , "Advanced Setting" , "Program Setting" , "Homing Setting"};

_Bool A_Now = 0 , A_Before = 0;

uint8_t settingIndex = 0;
uint8_t currentIndex = 0;
uint8_t accessLevel = 0;

uint16_t EncoderCounter = 0;
```

۶-۲-۳. معرفی تابع ControllerSetup

این تابع برای راه اندازی و پیکربندی های اولیه کنترلر تعبیه شده است، بدین صورت با روشن شدن کنترلر ابتدا این تابع اجرا شده و پارامترهای مهم برنامه مقداردهی اولیه می شوند. در ادامه محتویات موجود در این تابع را به ترتیب معرفی می نماییم.

دریافت و ذخیره اطلاعات از طریق سریال

```
HAL_UART_Receive_IT(&huart2, &rxByte, 1);
```

در ابتدا نیاز هست که تابع HAL_UART_Receive_IT را فراخوانی کنید، این تابع برای دریافت اطلاعات از طریق ارتباط سریال و ذخیره اطلاعات در بافر rxByte استفاده می شود. ورودی های این تابع بصورت زیر می باشد که huart* تعیین کننده کانال پورت سریال و pData* بافر دریافت بسته اطلاعاتی و Size نیز حجم بافر دریافتی یا تعداد خانه های آرایه ای که می خواهد دریافت شود می باشد.

HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)

پیکربندی و راه اندازی LCD

```
I2C_LCD_Init(16 , 2);
I2C_LCD_Backlight();
I2C_LCD_Display();
I2C_LCD_CreateCustomChar(0 , degree);
```

در این بخش تابع فعالسازی LCD را فراخوانی می کنیم که ورودی های این تابع به ترتیب ستون و سطرهای LCD را تعیین می کنند. در ادامه نیز توابع فعالسازی بک لایت و نمایشگر LCD فراخوانی می شوند تا در زمان راه اندازی LCD آماده بکار شود. درنهایت تابع ساخت یک کاراکتر خاص و سفارشی فراخوانی می شود، این تابع یک آرایه که از قبل توسط کاربر ساخته شده است را با شماره ردیف به عنوان ورودی دریافت می کند تا در زمانی که نیاز به استفاده از این کاراکتر باشد از آن استفاده کرد. لازم به ذکر است که این کاراکتر سفارشی یک علامت درجه است که در کتابخانه Program_Setting برای نمایش زاویه تنظیمی مفاصل ربات استفاده می شود.

نمایش پیام خوش آمدگویی

```
I2C_LCD_WriteString(0 , 0 , "SCARA CONTROLLER");
I2C_LCD_WriteString(1 , 1 , ">> WELCOME <<");
```

در این بخش با استفاده از تابع I2C_LCD_WriteString یک پیام خوش آمدگویی بر روی LCD نمایش می دهیم. ورودی های این تابع بصورت زیر می باشد که Col شماره ستون LCD می باشد که می تواند از عدد ۰ تا ۱۵ تنظیم شود و همچنین Row نیز شماره سطر LCD می باشد که می تواند بین عدد ۰ و ۱ تنظیم شود و درنهایت Str* رشته یا آرایه رشته مورد نظر می باشد که میخواهید آنرا بر روی LCD در مکان مورد نظر نمایش دهید.

void I2C_LCD_WriteString(uint8_t Col, uint8_t Row, char *Str)

➢ مقداردهی اولیه پارامترهای سرعت و پالس بر دور

```
for(uint8_t i = 0 ; i < 3 ; i++){
  speedValue[i] = 500.0;
  Pulse_rev[i] = 3200;
}
```

در این بخش یک حلقه for ایجاد می کنیم تا بتوانیم آرایه های سرعت و پالس بر دور را مقداردهی پیشفرض نماییم. همانطور که باید بدانید پارامترهای سرعت و پالس بر دور، برای سه استپرموتور تنظیم می شود پس برای ذخیره سازی مقادیر آنها از آرایه استفاده شده است و برای اینکه بتوان بصورت یکسان برای همه درایه ها یک مقدار تنظیم نمود باید حتما از حلقه for استفاده نماییم، بدین صورت برای پارامتر سرعت مقدار پیشفرض ۵۰۰ پالس بر ثانیه و برای پارامتر پالس بر دور نیز مقدار پیشفرض ۳۲۰۰ پالس تنظیم می شود.

➢ مقداردهی اولیه موقعیت های اولیه و ثانویه

```
xPosition[0] = 500.0;
yPosition[0] = 0.0;
```

برای اینکه بتوانیم مسیریابی خطی بین موقعیت های ربات داشته باشیم نیاز هست که موقعیت اولیه و ثانویه محورهای X و Y را در اختیار داشته باشیم، موقعیت ثانویه همیشه توسط کاربر تنظیم می شود و موقعیت اولیه نیز زمانی که موقعیت جدیدی ثبت می شود با موقعیت قبلی جایگزین خواهد شد اما زمانیکه هنوز موقعیتی توسط کاربر به ثبت نرسیده باید مقدار موقعیت اولیه بصورت پیشفرض تنظیم شود که در این دو آرایه مختصات قرارگیری پیشفرض بازوهای ربات تنظیم شده است.

➢ مقداردهی اولیه شمارنده انکودر

```
EncoderCounter = 0;

HAL_Delay(3000);
I2C_LCD_Clear();
```

مقدار متغیر EncoderCounter را برابر صفر قرار می دهیم که اگر تا قبل از آن دارای مقدار باشد، مقدار آن به حالت اولیه خود بازگردد. سپس یک تاخیر ۳ ثانیه ای ایجاد می کنیم تا پیام خوش آمدگویی به این مدت بر روی صفحه LCD نمایش داده شود و بعد از آن توسط دستور پاکسازی صفحه LCD را برای نمایش منوی اصلی آماده سازی می نماییم.

۶-۲-۴. معرفی تابع ControllerSetting

این تابع برای منوی اصلی کنترلر تعبیه شده است، بطوریکه با استفاده از این تابع می توانید منوی اصلی را بر روی LCD نمایش دهید و بواسطه دکمه های روی کنترلر که قبلا درمورد آنها صحبت شده بود وارد تنظیمات هر بخش شد و پارامترهای موجود در هر کدام را تنظیم و تغییر داد.

- **محدودسازی شمارنده انکودر برای جابجایی بین پارامترهای منو**

```
settingIndex = EncoderCounter % 4;
```

بطور کلی در همه توابعی که به عنوان یک منو استفاده می شوند، متغیری تعریف شده است که مقدار EncoderCounter را متناسب با تعداد پارامترهای منو محدود و در خود ذخیره می کند، بدین وسیله می توانید از این متغیر برای جابجایی بین منوها استفاده کنید. در اینجا از متغیر settingIndex استفاده شده است و همانطور که گفتیم مقدار EncoderCounter بدلیل حافظه ای که برای این متغیر درنظر گرفته شده است باید بین یک بازه عددی مورد نیاز محدود شود چراکه مقدار EncoderCounter می تواند از عدد ۰ تا ۶۵۵۳۵ تغییر کند و از آنجایی که منوی اصلی کنترلر شامل چهار بخش تنظیمات می باشد باید مقدار settingIndex بین عدد ۰ تا ۳ تغییر کند پس برای اینکار می توانیم از عملگر ٪ استفاده نماییم، این عملگر در زبان برنامه نویسی باقی مانده حاصل تقسیم را محاسبه و در اختیار کاربر قرار می دهد. پس بدین صورت می توانیم با محاسبه باقی مانده حاصل تقسیم EncoderCounter بر عدد مورد نظر، مقدار settingIndex را در هر بازه دلخواهی محدود نماییم.

- **پاکسازی صفحه LCD در هر جابجایی بین پارامترهای منو**

```
if(settingIndex != currentIndex){
  I2C_LCD_Clear();
  currentIndex = settingIndex;
}
```

برای جابجایی بین پارامترهای منو نیاز است که متن قبلی از روی صفحه LCD پاک شود تا بتوانیم متن بعدی را نمایش دهیم برای اینکار از یک دستور if استفاده شده است، بدین صورت زمانیکه مقدار settingIndex مخالف currentIndex بود، صفحه LCD پاک شود درنهایت مقدار settingIndex را با متغیر currentIndex برابر می کنیم تا فقط زمانی که تغییرات داشتیم اینکار انجام شود.

➢ نمایش پارامترهای منو بر روی صفحه LCD

```
I2C_LCD_WriteString(0 , 0 , settingMenu[settingIndex]);
I2C_LCD_WriteString(0 , 1 , "<Back      Next>");
```

در این بخش باید رشته های آرایه settingMenu که در بخش متغیرهای داخلی تعریف کردیم را بر روی LCD نمایش دهیم، برای اینکار می توانیم از تابع I2C_LCD_WriteString استفاده کنیم. بدین صورت متغیر settingIndex به عنوان اندیس آرایه settingMenu قرار می گیرد تا با هر بار چرخش روتاری انکودر منوی جدید نمایش داده شود.

➢ فراخوانی توابع دستورات کنترلی از وب سرور

```
HomingFromServer();
RunMoveFromServer();
RunCodeFromServer();
```

در این بخش سه تابع فراخوانی می شود که بوسیله آنها میتوان دستورات کنترلی ربات را از وب سرور دریافت کرد. توضیحات بیشتر این توابع را در کتابخانه Program_Setting داده خواهد شد.

➢ نحوه وارد شدن به هر بخش منوی اصلی

```
if(Enter_Button == 0){
  HAL_Delay(150);
  EncoderCounter = 0;
  accessLevel = 1;
  I2C_LCD_Clear();
  while(accessLevel == 1){
    switch(settingIndex){
      case 0 : BasicSetting(); break;

      case 1 : AdvancedSetting(); break;

      case 2 : ProgramSetting(); break;

      case 3 : HomingSetting(); break;
    }
  }
}
```

بعد از اینکه توانستیم بین منوها با روتاری انکودر جابجا شویم حال نوبت آن است که وارد هر یک از این منوها شد، برای اینکار باید طبق گفته های پیشین از دکمه فشاری روتاری انکودر استفاده نمود. بدین صورت با دستور if وضعیت این دکمه یعنی Enter_Button بررسی می شود که چون ورودی ها بصورت پول آپ به میکروکنترلر متصل شده اند پس باید مقدار Enter_Button به صفر منطقی تغییر یابد،

پس از تغییر یا فشرده شدن دکمه روتاری انکودر ابتدا یک تاخیر برای گرفتن دیبانس دکمه در نظر گرفته می شود و پس از آن باید مقدار EncoderCounter به حالت اولیه خود بازگردد. درنهایت باید عدد ۱ را در متغیری تحت عنوان accessLevel به معنای سطح دسترسی قرار دهیم؛ بطورکلی برای طبقه بندی شدن منوها، ما از الگوریتمی استفاده کردیم که برای دسترسی به طبقه زیرین هر منو باید ابتدا یک مقدار متناسب با هر طبقه به متغیر accessLevel اختصاص داده شود. بدین صورت برای وارد شدن به طبقه زیرین منو از یک حلقه while استفاده شده است که شرط ورود آن نیز مقدار accessLevel می باشد، این مقدار نیز همانطور که گفتیم متناسب با هر طبقه در نظر گرفته می شود، بطور مثال اگر در منوی اصلی قرار داشته باشیم و بخواهیم وارد منوی بعدی یعنی داخل منوی مورد نظر شویم نیاز هست که عدد ۱ را به این متغیر اختصاص دهیم و به همین شکل اگر زیر منویی وجود داشته باشد به ترتیب اعداد ۲ ، ۳ و ۴ شماره گذاری می شود. اینکار باعث می شود تا زمانیکه شرط accessLevel برقرار باشد نتوان از حلقه while خارج شد و همچنین برای خارج شدن از این حلقه نیز که در ادامه توضیح خواهیم داد باید مقدار accessLevel را به مقدار قبلی آن تغییر داد. حال پس از وارد شدن به این حلقه، میتوان بوسیله دستور سوئیچ کیس تعیین کرد که متناسب با مقدار settingIndex وارد کدام منو شوید. که در ادامه توابع این منوها را نیز بررسی خواهیم کرد.

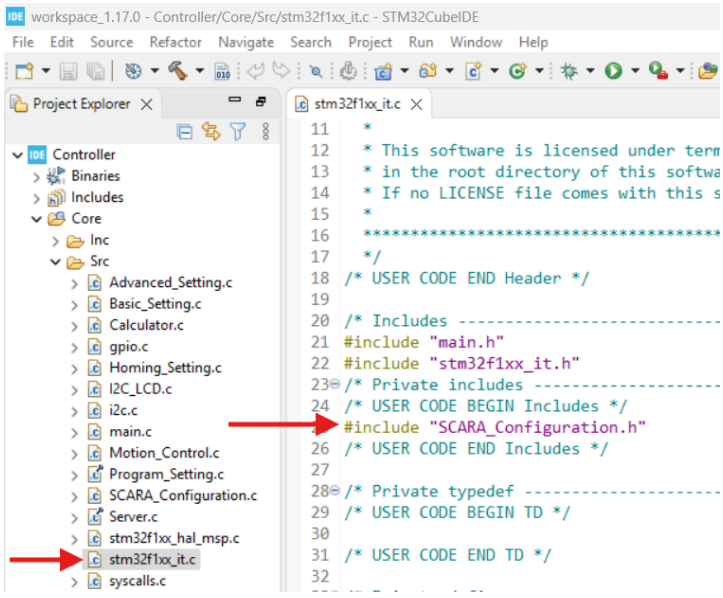
۶-۲-۵. معرفی تابع ReadEncoder

این تابع برای خواندن تعداد پالس های شمارش شده توسط روتاری انکودر تعبیه شده است، بطوریکه با چرخش روتاری انکودر به چپ و راست میتوان مقدار متغیر EncoderCounter را افزایش و کاهش داد.

```
void ReadEncoder(void){
  A_Now = Encoder_A;

  if(A_Now != A_Before){
    if(Encoder_B != A_Now) EncoderCounter++;
    else EncoderCounter--;
    A_Before = A_Now;
  }
}
```

در این تابع دو متغیر با نام های A_Now و A_Before وجود دارند که وضعیت پایه A روتاری انکودر را در دو حالت قبل و بعد می خوانند. همانطور که در فصل قبل در بخش معرفی روتاری انکودر توضیح داده شده بود، وضعیت پایه های A و B روتاری انکودر در هر چرخش بواسطه پله هایی که در ساختار داخلی آن وجود دارد، بصورت صفر و یک تغییر می کنند یعنی پالس هایی با فاصله زمانی مشخص تولید می شوند که این پالس های تولید شده از هر پایه روتاری انکودر با یکدیگر ۹۰ درجه اختلاف فاز الکتریکی دارند، بدین صورت میتوان جهت چرخش انکودر را تشخیص داد. پس در نتیجه برای اینکه بتوان

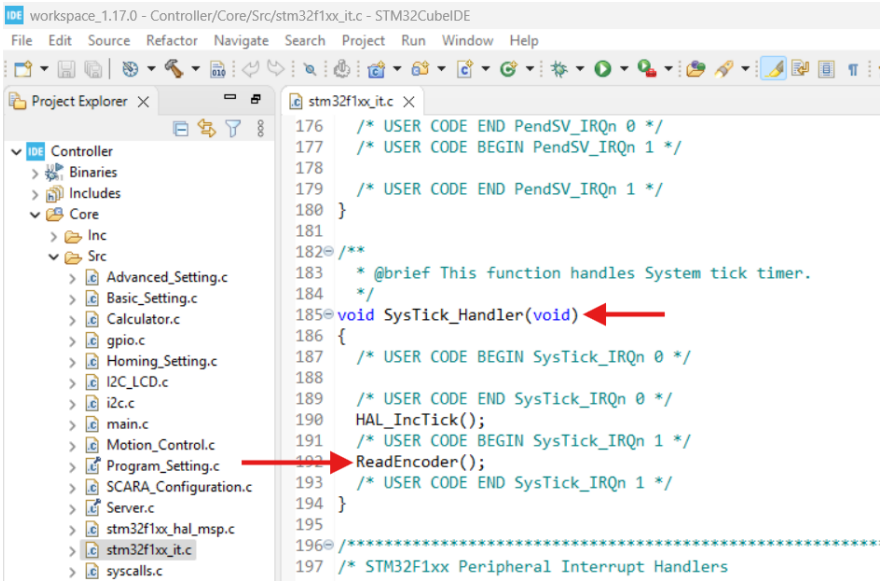
تشخیص داد که انکودر درحال چرخش است باید وضعیت قبل و بعد آنرا در یک متغیر ذخیره کنیم. حال پس از دانستن وضعیت قبل و بعد پایه A میتوان با یک دستور if تعیین نمود که آیا انکودر چرخیده است یا خیر، بدین صورت حتما باید دو وضعیت پایه A با یکدیگر نامساوی باشند یعنی زمانی که وضعیت این پایه از مقدار صفر منطقی به یک منطقی تغییر کند نیاز داریم که مقدار متغیر EncoderCounter را افزایش یا کاهش دهیم. برای اینکه جهت چرخش را نیز تشخیص دهیم همانطور که گفتیم زمانی که وضعیت پایه A و B برابر نباشد یعنی روتاری انکودر در جهت عقربه های ساعت چرخانده شده و مقدار متغیر EncoderCounter باید افزایش یابد و زمانی که وضعیت پایه A و B برابر باشد یعنی روتاری انکودر در جهت عکس عقربه های ساعت چرخانده شده و مقدار متغیر EncoderCounter باید کاهش یابد. اینکار با یک دستور if دیگر در داخل دستور if قبلی انجام می شود، بدین صورت که وضعیت Encoder_B که قبلا به عنوان ماکرو تعریف شده بود با وضعیت A_Now مقایسه می شود. در نهایت نیز مقدار A_Now در A_Before ریخته می شود تا این تابع برای چرخش بعدی روتاری انکودر آماده باشد. این تابع درنهایت در کتابخانه stm32f1xx_it فراخوانی خواهد شد. همانطور که در شکل ۶-۲۹ مشاهده می نمایید ابتدا باید کتابخانه SCARA_Configuration را در این مکان قرار دهیم تا بتوانیم تابع ReadEncoder را فراخوانی نماییم.



شکل ۶- ۲۹- فراخوانی کتابخانه SCARA_Configuration

همانطور که در شکل ۶-۳۰ مشاهده می کنید، تابع ReadEncoder در تابع SysTick_Handler فراخوانی شده است اما چرا؟ برای اینکه بتوانیم وضعیت پایه های A و B روتاری انکودر را در هر لحظه از میکروکنترلر بدون هیچ تاخیری بخوانیم باید از توابعی استفاده کنیم که واحد وقفه برای آنها فعالسازی

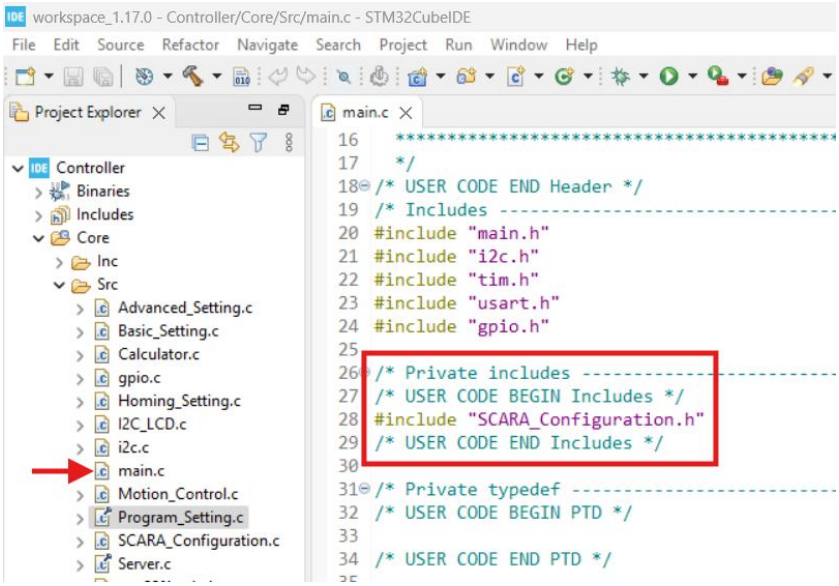
شده است. از آنجایی که تابع ReadEncoder یک تابعی است که واحد فقه برای آن فعالسازی نشده است نیاز داریم که آنرا در یک تابع دیگر که واحد وقفه برای آن فعالسازی شده فراخوانی نماییم. البته دقت شود که این روش استانداردی نیست و ممکن است در طولانی مدت بر عملکرد سیستم تاثیر بگذارد بنابراین توصیه می شود که برای اینکار حتما واحد وقفه را برای یکی از ورودی ها انکودر فعالسازی کنید و از تابع مخصوص آن استفاده نمایید.



شکل ۶- ۳۰- فراخوانی تابع ReadEncoder

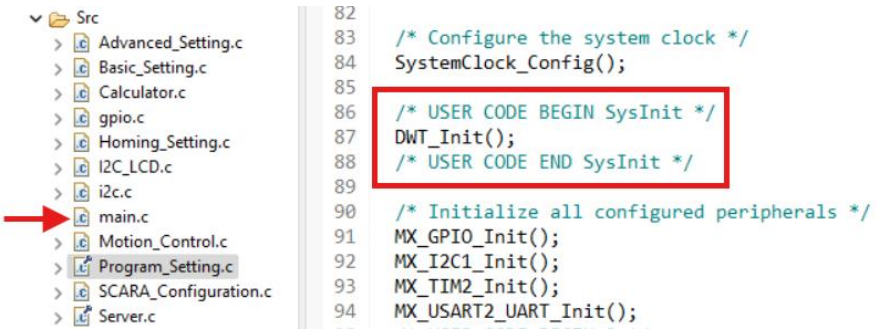
۶-۲-۶. معرفی تابع Main

این تابع، تابع اصلی پروژه شناخته می شود که درنهایت تمامی کتابخانه ها و توابع موجود در پروژه باید در این تابع فراخوانی شوند تا در زمان کامپایل شدن، کدها به زبان ماشین تبدیل شوند و شما بتوانید برنامه را بر روی میکروکنترلر خود دانلود نمایید. کدهایی که در این تابع قرار می گیرند به شرح زیر است.



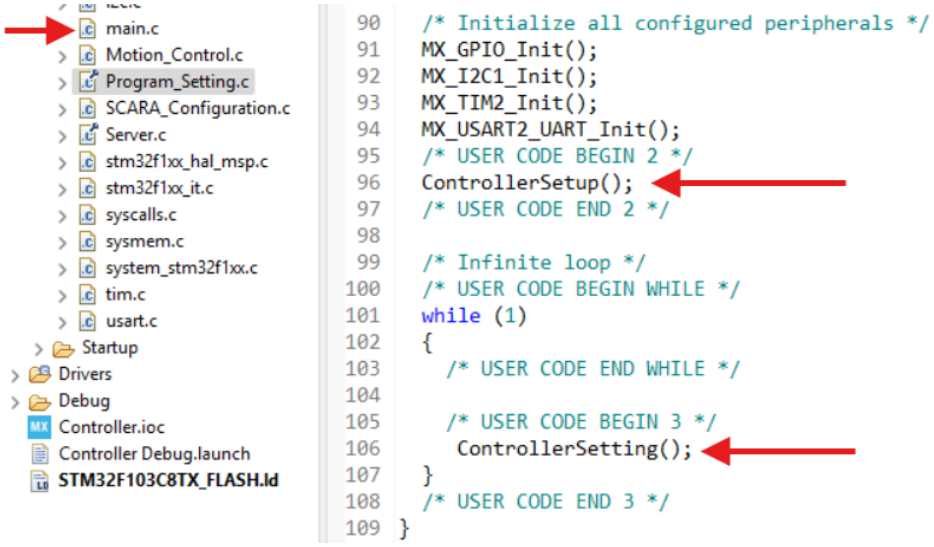
شکل ۶- ۳۱- فراخوانی کتابخانه SCARA_Configuration در تابع اصلی

در ابتدا کتابخانه SCARA_Configuration که قبلا ایجاد کرده بودیم را در این مکان فراخوانی می نماییم و سپس در ادامه تابع فعالساز DWT[1] را در محل علامت زده شده شکل ۶-۳۲ قرار می دهیم، این تابع فعالسازی برای کتابخانه Micros می باشد که بصورت دستی به پروژه اضافه شده است. از آنجایی که توابع هال فقط تاخیرها و زمان سنج ها را برحسب میلی ثانیه ارائه می دهند برای استفاده از تاخیرهایی برحسب میکروثانیه باید آنها را بصورت دستی ساخت این کتابخانه نیز به همین منظور به پروژه اضافه شده است.



شکل ۶- ۳۲- جایگذاری تابع فعالساز DWT در تابع اصلی

درنهایت توابع ControllerSetup و ControllerSetting نیز به ترتیب در مکان های علامت زده شده در شکل ۶-۳۳ فراخوانی می شوند تا برنامه بدرستی اجرا شود.



شکل ۶- ۳۳- فراخوانی توابع ControllerSetup و ControllerSetting

[1] Debug Watchpoint & Trace

۶-۳. معرفی و بررسی کتابخانه Server

این کتابخانه برای تبادل اطلاعات بصورت دوطرفه همزمان بین کنترلر و صفحه وب ایجاد شده است، در این کتابخانه توابعی وجود دارد که با استفاده از آنها می توانید موقعیت های تنظیم شده ربات را کنترل و پارامترهای هر تنظیمات را به صفحه وب ارسال یا از صفحه وب دریافت نمایید. این توابع نیز در هدر فایل تعریف شده اند تا از آنها در سورس فایل، برای نوشتن کدهای اصلی برنامه استفاده نماییم.

```
//-----[ Prototypes For User External Functions ]-----
float extractNumFromCode(char *code, char *character);

void reseiveSetting(char *code);
void reseiveMove(char *code);
void reseiveCode(char *code);

void sendProgram(float x , float y , float z , float t);
void sendSetting(int setNum , int value);
void sendRun(void);
void sendPause(void);
void sendClear(void);
void sendOk(void);
```

۶-۳-۱. معرفی متغیرهای خارجی

متغیرهای خارجی، متغیرهایی هستند که در کتابخانه های دیگر تعریف و مقداردهی اولیه شدند و در اینجا برای خواندن و تغییر مقدار آنها استفاده می شوند. این متغیرها بوسیله دستور extern به این کتابخانه فراخوانی می شوند.

```
/*-----------------------[EXTERNAL VARIABLES]-----------------------*/
extern UART_HandleTypeDef huart2;

extern _Bool kinematicType;

extern uint16_t PositionCounter;
extern uint16_t Pulse_rev[3];
extern uint16_t Pulse_revMenu[4];
extern uint16_t indexCodeState;

extern float speedValue[3];
extern float xPosition[2] , yPosition[2];
extern float jointLength_1 , jointLength_2;
```

۶-۳-۲. معرفی متغیرهای داخلی

متغیرهای داخلی، متغیرهایی هستند که در این کتابخانه تعریف و مقداردهی اولیه شدند. این متغیرها می توانند در این کتابخانه یا کتابخانه های دیگر نیز استفاده شوند.

```
/*-----------------------[INTERNAL VARIABLES]-----------------------*/
char buffer[50];
char rxBuffer[100];

bool moveMode = false;
bool runPosition = false;
bool runCode = false;
bool homeState[4];

uint8_t rxByte;

uint16_t rxIndex = 0;
uint16_t indexCode = 0;
```

۶-۳-۳. معرفی تابع extractNumFromCode

این تابع برای استخراج یک عدد اعشاری از یک رشته کد استفاده می‌شود. اینکار را بر اساس یک کاراکترکه به عنوان ورودی داده می‌شود انجام می‌دهد. به زبان ساده، این تابع به دنبال یک کاراکتر در رشته مورد نظر می‌گردد و هر چیزی که بعد از آن کاراکتر قرار دارد را به عدد اعشاری تبدیل می‌کند و برمی‌گرداند.

```
float extractNumFromCode(char *code, char *character) {
  char *ptr = strstr(code, character);

  if(ptr != NULL) {
    ptr += strlen(character);
    return atof(ptr);
  }
  return NAN;
}
```

در ابتدا، تابع strstr به دنبال اولین تطابق character در رشته code می‌گردد و آدرس آنرا برمی‌گرداند. این آدرس در متغیر ptr ذخیره می‌شود. اگر متغیر ptr برابر NULL نباشد (یعنی character در code پیدا شود)، متغیر ptr را به اندازه طول character جلو می‌بریم تا به اولین کاراکتر بعد از character برسیم. سپس، تابع atof [1] مقدار عددی را که از موقعیت فعلی متغیر ptr شروع می‌شود به عدد اعشاری[2] تبدیل می‌کند و آنرا برمی‌گرداند. اگر متغیر ptr برابر NULL باشد (یعنی character در code پیدا نشود)، تابع مقدار[3] NAN را برمی‌گرداند. بطور مثال فرض کنید متغیر code برابر با "G0 X225.5 Y125.5" و متغیر character برابر با "X" باشد. تابع ابتدا به دنبال کاراکتر "X" می‌گردد و وقتی آن را پیدا کرد، اشاره‌گر را به بعد از "X" جلو می‌برد و از موقعیت "225.5" شروع به تبدیل می‌کند که نتیجه نهایی عدد 225.5 به صورت float خواهد بود.

[1] ASCII to float
[2] float
[3] Not A Number

۶-۳-۴. معرفی تابع HAL_UART_RxCpltCallback

این تابع درواقع وظیفه دارد تا بایت‌های دریافتی از طریق UART را تجزیه و تحلیل کند و بر اساس محتوای آن‌ها اقدامات مختلفی را انجام دهد. این اقدامات می‌تواند شامل پردازش تنظیمات، حرکت‌ها، پایان عملیات، دریافت کدها و افزایش شمارنده موقعیت باشد. ورودی این تابع بصورت زیر می باشد که *huart تعیین کننده کانال پورت سریال می باشد.

HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)

- **بررسی پایان خط بایت دریافتی**

```
if(rxByte == '\n'){
  rxBuffer[rxIndex] = '\0';
  rxIndex = 0;
```

اگر بایت دریافتی \n باشد، به معنای پایان خط است. اینجا متغیر rxBuffer را با کاراکتر پایان \0 کامل می‌کند و rxIndex را صفر می‌کند تا از اول شروع کند.

- **بررسی محتوای پیام دریافتی**

```
if(strstr(rxBuffer, "setting") != NULL) reseiveSetting(rxBuffer);
```

اگر پیام حاوی "setting" باشد، تابع reseiveSetting با محتوای rxBuffer فراخوانی می‌شود تا تنظیمات مربوطه پردازش شوند.

```
else if(strstr(rxBuffer, "Move") != NULL) reseiveMove(rxBuffer);
```

اگر پیام حاوی "Move" باشد، تابع reseiveMove با محتوای rxBuffer فراخوانی می‌شود تا حرکت مورد نظر پردازش شود.

```
else if(strstr(rxBuffer, "End") != NULL){
  runPosition = false;
  runCode = false;
  indexCode = 0;
}
```

اگر پیام حاوی "End" باشد، متغیرهای runPosition و runCode غیر فعال می‌شوند (false شده) و متغیر indexCode نیز صفر می‌شود.

```
else if(strstr(rxBuffer, "Position") != NULL || strstr(rxBuffer, "Code") != NULL){
  reseiveCode(rxBuffer);
}
```

اگر پیام حاوی "Position" یا "Code" باشد، تابع reseiveCode با محتوای rxBuffer فراخوانی می‌شود تا کد مورد نظر پردازش شود.

```
else if(strstr(rxBuffer, "++") != NULL) PositionCounter++;
```

اگر پیام حاوی "++" باشد، شمارنده PositionCounter افزایش می‌یابد.

- **اضافه کردن بایت دریافتی به بافر (در صورت عدم پایان خط)**

```
else {
  if(rxIndex < sizeof(rxBuffer) - 1) rxBuffer[rxIndex++] = rxByte;
}
```

اگر بایت دریافتی \n نباشد، بایت دریافتی به rxBuffer اضافه می‌شود و rxIndex افزایش می‌یابد.

۶-۳-۵. معرفی تابع reseiveSetting

این تابع به منظور پردازش دستورات تنظیماتی است که از طریق UART دریافت می‌شود. بر اساس دستورات دریافتی، این تابع تنظیمات مختلفی را اعمال می‌کند مانند فعال کردن حالت‌های مختلف، تنظیم سرعت استپر موتورها، تعیین نوع سینماتیک، تنظیم تعداد پالس‌های موتورها و طول مفاصل. این تابع یک پارامتر ورودی به نام code دارد. این پارامتر شامل دستورات تنظیماتی است که باید پردازش شوند.

void reseiveSetting(char *code)

- **بررسی وضعیت AutoHoming**

```
if(strstr(rxBuffer, "AutoHoming") != NULL) homeState[0] = true;
```

در اینجا تابع strstr بررسی می‌کند که آیا متغیر rxBuffer حاوی رشته "AutoHoming" است یا خیر. اگر بله، مقدار homeState[0] را به true تنظیم می‌کند و عملیات هومینگ بصورت خودکار برای همه محورها انجام می شود.

- **بررسی وضعیت Home Joint1**

```
else if(strstr(rxBuffer, "Joint1") != NULL) homeState[1] = true;
```

در اینجا تابع strstr بررسی می‌کند که آیا متغیر rxBuffer حاوی رشته "Joint1 " است یا خیر. اگر بله، مقدار homeState[1] را به true تنظیم می‌کند و عملیات هومینگ بصورت دستی برای محور اول انجام می شود.

➢ بررسی وضعیت Home Joint2

```
else if(strstr(rxBuffer, "Joint2") != NULL) homeState[2] = true;
```

در اینجا تابع strstr بررسی می‌کند که آیا متغیر rxBuffer حاوی رشته "Joint2 " است یا خیر. اگر بله، مقدار homeState[2] را به true تنظیم می‌کند و عملیات هومینگ بصورت دستی برای محور دوم انجام می شود.

➢ بررسی وضعیت Home Z

```
else if(strstr(rxBuffer, "Joint3") != NULL) homeState[3] = true;
```

در اینجا تابع strstr بررسی می‌کند که آیا متغیر rxBuffer حاوی رشته "Joint3 " است یا خیر. اگر بله، مقدار homeState[3] را به true تنظیم می‌کند و عملیات هومینگ بصورت دستی برای محور سوم انجام می شود.

➢ پردازش سایر دستورات تنظیمات

```
else {
  uint8_t setNum = extractNumFromCode(code, "P");
  uint16_t value = extractNumFromCode(code, "=");
```

اگر متغیر rxBuffer حاوی هیچ یک از رشته‌های بالا نباشد، تابع به بخش دیگری از کد می‌رود. در اینجا، از دو تابع به نام extractNumFromCode برای استخراج اعداد از متغیر code استفاده می‌شود:

- متغیر setNum : شماره پارامتر که با استفاده از کاراکتر "P" استخراج می‌شود.
- متغیر value : مقدار پارامتر که با استفاده از کاراکتر "=" استخراج می‌شود.

اعمال تنظیمات بر اساس شماره پارامتر

```
switch(setNum){
  case 0 : for(uint8_t i = 0 ; i < 3 ; i++) speedValue[i] = value; break;

  case 1 : kinematicType = value; break;

  case 2 : for(uint8_t i = 0 ; i < 3 ; i++) Pulse_rev[i] = Pulse_revMenu[value]; break;

  case 3 : speedValue[0] = value; break;

  case 4 : Pulse_rev[0] = Pulse_revMenu[value]; break;

  case 5 : speedValue[1] = value; break;

  case 6 : Pulse_rev[1] = Pulse_revMenu[value]; break;

  case 7 : speedValue[2] = value; break;

  case 8 : Pulse_rev[2] = Pulse_revMenu[value]; break;

  case 9 : jointLength_1 = value; break;

  case 10 : jointLength_2 = value; break;
}
```

این بخش از کد با استفاده از دستور سوئیچ کیس بر اساس مقدار setNum، تنظیمات مختلفی را اعمال می شود:

- اگر مقدار setNum برابر 0 باشد، پارامتر سرعت برای همه استپر موتورها تنظیم می شود.
- اگر مقدار setNum برابر 1 باشد، نوع سینماتیک (مستقیم یا معکوس) تنظیم می شود.
- اگر مقدار setNum برابر 2 باشد، پارامتر پالس بر دور برای همه استپر موتورها تنظیم می شود.
- اگر مقدار setNum برابر 3 باشد، پارامتر سرعت برای استپر موتور ۱ تنظیم می شود.
- اگر مقدار setNum برابر 4 باشد، پارامتر پالس بر دور برای استپر موتور ۱ تنظیم می شود.
- اگر مقدار setNum برابر 5 باشد، پارامتر سرعت برای استپر موتور ۲ تنظیم می شود.
- اگر مقدار setNum برابر 6 باشد، پارامتر پالس بر دور برای استپر موتور ۲ تنظیم می شود.
- اگر مقدار setNum برابر 7 باشد، پارامتر سرعت برای استپر موتور ۳ تنظیم می شود.
- اگر مقدار setNum برابر 8 باشد، پارامتر پالس بر دور برای استپر موتور ۳ تنظیم می شود.
- اگر مقدار setNum برابر 9 باشد، حداکثر طول بازوی اول تنظیم می شود.
- اگر مقدار setNum برابر 10 باشد، حداکثر طول بازوی دوم تنظیم می شود.

۶-۳-۶. معرفی تابع reseiveMove

این تابع برای پردازش دستورات حرکتی استفاده می‌شود. در این تابع، مختصات X ، Y و Z از دستور دریافتی استخراج شده و تنظیمات مربوط به موقعیت‌ها به‌روزرسانی می‌شوند. این تابع دارای یک پارامتر ورودی به نام code می باشد. این پارامتر شامل دستور حرکتی است که از طریق UART دریافت می شود.
void reseiveMove(char *code)

- **تعریف متغیرهای محلی و فعال‌سازی حالت Move**

```
float x , y , z;
moveMode = true;
```

در اینجا، سه متغیر x ، y و z از نوع float تعریف می‌شوند که برای ذخیره مختصات X، Y و Z استفاده می‌شوند. همچنین متغیر moveMode به true تنظیم می‌شود که نشان می‌دهد ربات می تواند با دریافت دستور Move به موقعیت مورد نظر حرکت کند.

- **استخراج مختصات X و Y و Z**

```
x = extractNumFromCode(code, "X");
y = extractNumFromCode(code, "Y");
z = extractNumFromCode(code, "Z");
```

در اینجا از تابع extractNumFromCode برای استخراج مقادیر مربوط به مختصات X ، Y و Z از رشته code استفاده می‌شود. این تابع به دنبال کاراکترهای X ، Y و Z می‌گردد و اعداد بعد از آنها را استخراج می‌کند.

- **به‌روزرسانی موقعیت‌های ثانویه**

```
xPosition[1] = x;
yPosition[1] = y;
```

در این بخش، مقدار x به xPosition[1] و مقدار y به yPosition[1] اختصاص داده می‌شود. این مقادیر برای تنظیم موقعیت جدید ربات استفاده می‌شوند.

۶-۳-۷. معرفی تابع reseiveCode

این تابع برای پردازش دستورات مختلفی استفاده می‌شود که ممکن است شامل پاک کردن موقعیت‌ها، توقف عملیات، تنظیم حالت‌های حرکتی و بروزرسانی مختصات باشد. این تابع بر اساس دستور دریافتی، اقدامات متفاوتی انجام می‌دهد. این تابع دارای یک پارامتر ورودی به نام code می باشد. این پارامتر شامل دستور کد است که از طریق UART دریافت می شود.

void reseiveCode(char *code)

- **تعریف متغیرهای محلی**

```
float x , y , z;
```

در اینجا سه متغیر x ، y و z از نوع float تعریف شده‌اند که برای ذخیره مختصات استفاده می‌شوند.

- **بررسی وضعیت پاکسازی موقعیت‌ها**

```
if(strstr(rxBuffer, "Positions Clear") != NULL) PositionCounter = 0;
```

در اینجا تابع strstr بررسی می‌کند که آیا rxBuffer حاوی رشته "Positions Clear" است یا خیر. اگر بله، مقدار PositionCounter به صفر تنظیم می‌شود که به معنای پاک کردن تمام موقعیت‌ها است.

- **بررسی وضعیت توقف برنامه**

```
else if(strstr(rxBuffer, "Pause") != NULL){
  indexCodeState = 0;
  runPosition = false;
  runCode = false;
}
```

این بخش بررسی می‌کند که آیا rxBuffer حاوی رشته "Pause" است یا خیر. اگر بله، مقدار متغیر indexCodeState به صفر تنظیم می شود و مقادیر runPosition و runCode به false تنظیم می‌شوند که به معنای توقف عملیات است.

- **بررسی وضعیت بازگشت به موقعیت قبلی**

```
else if(strstr(rxBuffer, "pBack") != NULL){
  indexCodeState = 0;
  runPosition = false;
  runCode = false;
}
```

این بخش بررسی می‌کند که آیا rxBuffer حاوی رشته "pBack" است یا خیر. اگر بله، مقدار متغیر indexCodeState به صفر تنظیم می شود و مقادیر runPosition و runCode به false تنظیم می‌شوند که به معنای بازگشت به حالت قبلی است.

➢ بررسی و تنظیم حالت Position

```
if(strstr(rxBuffer, "Position") != NULL){
  if(runPosition == false) runPosition = true;
}
```

اگر rxBuffer حاوی رشته "Position" باشد و متغیر runPosition به false تنظیم شده باشد، آنرا به true تنظیم می‌کند. این به معنای فعال کردن حالت حرکت Position است.

➢ بررسی و تنظیم حالت Code

```
else if(strstr(rxBuffer, "Code") != NULL){
  if(runCode == false) runCode = true;
}
```

اگر rxBuffer حاوی رشته "Code" باشد و متغیر runCode به false تنظیم شده باشد، آنرا به true تنظیم می‌کند. این به معنای فعال کردن حالت حرکت Code است.

➢ استخراج مختصات X و Y و Z

```
indexCode = extractNumFromCode(code, "I");
x = extractNumFromCode(code, "X");
y = extractNumFromCode(code, "Y");
z = extractNumFromCode(code, "Z");
```

در اینجا از تابع extractNumFromCode برای استخراج مقادیر مربوط به مختصات X ، Y ، Z و indexCode از رشته code استفاده می‌شود. این تابع به دنبال کاراکترهای X ، Y ، Z و I می‌گردد و اعداد بعد از آنها را استخراج می‌کند.

➢ به‌روزرسانی موقعیت‌های ثانویه

```
if(!isnan(x)) xPosition[1] = x;
if(!isnan(y)) yPosition[1] = y;
```

در اینجا اگر مقادیر x و y دارای عدد باشند یا NaN نباشند، مقدار x به xPosition[1] و مقدار y به yPosition[1] اختصاص داده می‌شود. این مقادیر برای تنظیم موقعیت جدید ربات استفاده می‌شوند.

۶-۳-۸. معرفی تابع sendProgram

این تابع برای ارسال مختصات و زمان به وب سرور از طریق UART استفاده می‌شود. داده‌ها در قالب یک رشته فرمت‌دار تولید شده و از طریق UART ارسال می‌شوند. این تابع دارای چهار پارامتر ورودی x ، y ، z و t از نوع float می باشد.

void sendProgram(float x , float y , float z , float t)

➢ پاک کردن محتویات بافر

```
memset(buffer, 0, 50);
```

در اینجا از تابع memset برای پاک کردن محتویات buffer استفاده می‌شود و همه‌ی بایت‌های آنرا به صفر تنظیم می‌کند. حجم بافر ۵۰ بایت است. اینکار به منظور اطمینان از اینکه بافر برای ذخیره‌سازی رشته جدید آماده باشد انجام می‌شود.

➢ فرمت‌دهی به رشته و ذخیره در بافر

```
sprintf(buffer, "Positions I%d = X%.1f Y%.1f Z%.1f T%.1f\n",
PositionCounter + 1, x, y, z, t);
```

این خط از تابع sprintf برای ایجاد یک رشته فرمت‌دار استفاده می‌کند و آن را در buffer ذخیره می‌کند. رشته تولید شده شامل اطلاعات زیر می باشد.

- متغیر PositionCounter : شماره موقعیت فعلی
- متغیر x : مقدار مختصات X
- متغیر y : مقدار مختصات Y
- متغیر z : مقدار مختصات Z
- متغیر t : مقدار زمان T

به عنوان مثال، اگر x = 12.3 ، y = 45.6 ، z = 78.9 ، t = 250 و PositionCounter = 2 باشد، رشته تولید شده به شکل زیر خواهد بود.

Positions I3 = X12.3 Y45.6 Z78.9 T250

➢ **ارسال داده‌ها از طریق تابع HAL_UART_Transmit_IT**

HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));

این خط از تابع HAL_UART_Transmit_IT برای ارسال داده‌های موجود در buffer از طریق UART استفاده می‌کند. پارامترهای ورودی شامل کانال پورت سریال، بافر بسته اطلاعاتی و طول بافر که با استفاده از تابع strlen محاسبه می‌شود.

۶-۳-۹. معرفی تابع sendRun

این تابع برای ارسال پیام "Positions Run\n" به وب سرور از طریق UART استفاده می‌شود. این پیام می‌تواند به منظور اعلام شروع یا ادامه عملیات به وب سرور استفاده شود. ارسال این پیام با استفاده از واحد وقفه انجام می‌شود که به میکروکنترلر اجازه می‌دهد تا عملیات ارسال را بدون بلاک کردن پردازش‌های دیگر انجام دهد.

```
HAL_UART_Transmit_IT(&huart2, (uint8_t*)"Positions Run\n", 14);
```

این خط از تابع HAL_UART_Transmit_IT برای ارسال پیام "Positions Run\n" از طریق UART استفاده می‌کند. پارامترهای ورودی شامل کانال پورت سریال، رشته "Positions Run\n" و طول رشته "Positions Run\n" که شامل ۱۳ کاراکتر و یک کاراکتر پایان خط است که در مجموع ۱۴ بایت طول دارد؛ طول رشته به تابع اطلاع می‌دهد که چه مقدار داده باید ارسال شود.

۶-۳-۱۰. معرفی تابع sendPause

این تابع برای ارسال پیغام توقف یا Pause به وب سرور از طریق UART استفاده می‌شود. این تابع بسته به وضعیت فعلی (runPosition یا runCode)، پیغام مناسب را ارسال می‌کند و حالت‌ها و متغیرهای مربوطه را بروزرسانی می‌کند.

➢ **بررسی وضعیت runPosition**

```
if(runPosition){
  HAL_UART_Transmit_IT(&huart2, (uint8_t*)"Positions Pause\n", 16);
  runPosition = false;
  indexCodeState = 0;
}
```

این شرط بررسی می‌کند که آیا runPosition فعال است یعنی برابر با true است یا خیر. اگر این شرط برقرار باشد، به این معناست که ربات در حال اجرای برنامه است و دستور توقف برای وب سرور ارسال خواهد شد.

- **ارسال پیغام Positions Pause از طریق تابع HAL_UART_Transmit_IT**

```
HAL_UART_Transmit_IT(&huart2, (uint8_t*)"Positions Pause\n", 16);
```

اگر runPosition فعال باشد، این خط از تابع HAL_UART_Transmit_IT برای ارسال پیام "Positions Pause\n" از طریق UART استفاده می‌کند. پارامترهای ورودی شامل کانال پورت سریال، رشته "Positions Pause\n" و طول رشته "Positions Pause\n" که شامل ۱۵ کاراکتر و یک کاراکتر پایان خط است که در مجموع ۱۶ بایت طول دارد.

- **غیرفعال کردن runPosition و ریست کردن متغیر indexCodeState**

```
runPosition = false;
indexCodeState = 0;
```

در اینجا، مقدار runPosition به false تنظیم می‌شود که به معنای غیرفعال کردن حالت Position است. همچنین متغیر indexCodeState به صفر تنظیم می‌شود که نشان‌دهنده این است که برنامه ربات متوقف شده و در حالت اولیه قرار گرفته است.

- **بررسی وضعیت runCode**

```
else if(runCode){
  HAL_UART_Transmit_IT(&huart2, (uint8_t*)"Code Pause\n", 11);
  runCode = false;
  indexCodeState = 0;
}
```

اگر شرط runPosition برقرار نباشد، این شرط بررسی می‌کند که آیا runCode فعال است یعنی برابر با true است یا خیر. اگر این شرط برقرار باشد، به این معناست که ربات در حال اجرای برنامه می باشد و دستور توقف برای کدها ارسال خواهد شد.

➢ **ارسال پیغام Code Pause از طریق تابع HAL_UART_Transmit_IT**

```
HAL_UART_Transmit_IT(&huart2, (uint8_t*)"Code Pause\n", 11);
```

اگر runCode فعال باشد، این خط از تابع HAL_UART_Transmit_IT برای ارسال پیام "Code Pause\n" از طریق UART استفاده می‌کند. پارامترهای ورودی شامل کانال پورت سریال، رشته "Code Pause\n" و طول رشته "Code Pause\n" که شامل ۱۰ کاراکتر و یک کاراکتر پایان خط است که در مجموع ۱۱ بایت طول دارد.

➢ **غیرفعال کردن runCode و ریست کردن متغیر indexCodeState**

```
runCode = false;
indexCodeState = 0;
```

در اینجا، مقدار runCode به false تنظیم می‌شود که به معنای غیرفعال کردن حالت Code است. همچنین متغیر indexCodeState به صفر تنظیم می‌شود که نشان‌دهنده این است که برنامه ربات متوقف شده و در حالت اولیه قرار گرفته است.

۶-۳-۱۱. معرفی تابع sendClear

این تابع برای ارسال پیغام پاکسازی یا Clear به وب سرور از طریق UART استفاده می‌شود. همچنین، این تابع متغیر PositionCounter را به صفر تنظیم می‌کند تا تمام موقعیت‌های قبلی پاک شوند و شمارش از ابتدا شروع شود.

➢ **ارسال پیغام Positions Clear از طریق HAL_UART_Transmit_IT**

```
HAL_UART_Transmit_IT(&huart2, (uint8_t*)"Positions Clear\n", 16);
```

این خط از تابع HAL_UART_Transmit_IT برای ارسال پیام "Positions Clear\n" از طریق UART استفاده می‌کند. پارامترهای ورودی شامل کانال پورت سریال، رشته "Positions Clear\n" و طول رشته "Positions Clear\n" که شامل ۱۵ کاراکتر و یک کاراکتر پایان خط است که در مجموع ۱۶ بایت طول دارد.

➢ **ریست کردن شمارنده موقعیت**

```
PositionCounter = 0;
```

این خط مقدار PositionCounter را به صفر تنظیم می‌کند. این عمل به معنای پاکسازی تمام موقعیت‌های ثبت شده و شروع شمارش از ابتدا است.

۶-۳-۱۲. معرفی تابع sendOk

این تابع برای ارسال یک پیام "Ok" به وب سرور از طریق UART استفاده می‌شود. این پیام تاییدیه اتمام اجرای تمام موقعیت های ارسال شده از طرف وب سرور می باشد. ارسال این پیام با استفاده از واحد وقفه انجام می‌شود که به میکروکنترلر اجازه می‌دهد تا عملیات ارسال را بدون بلاک کردن پردازش‌های دیگر انجام دهد.

```
HAL_UART_Transmit_IT(&huart2, (uint8_t*)"Ok\n", 3);
```

این خط از تابع HAL_UART_Transmit_IT برای ارسال پیام "Ok\n" از طریق UART استفاده می‌کند. پارامترهای ورودی شامل کانال پورت سریال، رشته "Ok\n" و طول رشته "Ok\n" که شامل ۲ کاراکتر و یک کاراکتر پایان خط است که در مجموع ۳ بایت طول دارد.

۶-۳-۱۳. معرفی تابع sendSetting

این تابع برای ارسال تنظیمات خاص به وب سرور از طریق UART استفاده می‌شود. این تابع یک پیغام فرمت‌دار ایجاد می‌کند که شامل شماره تنظیم (setNum) و مقدار تنظیم (value) است و آنرا از طریق UART ارسال می‌کند. این تابع دارای دو پارامتر ورودی setNum و value از نوع int دارد.

- **پاک کردن محتویات بافر**

```
memset(buffer, 0, 50);
```

در اینجا از تابع memset برای پاک کردن محتویات buffer استفاده می‌شود و همه‌ی بایت‌های آنرا به صفر تنظیم می‌کند. حجم بافر ۵۰ بایت است. اینکار به منظور اطمینان از اینکه بافر برای ذخیره‌سازی رشته جدید آماده باشد انجام می‌شود.

- **فرمت‌دهی به رشته و ذخیره در بافر**

```
sprintf(buffer, "Setting P%d =%d\n", setNum, value);
```

این خط از تابع sprintf برای ایجاد یک رشته فرمت‌دار استفاده می‌کند و آن را در buffer ذخیره می‌کند. رشته تولید شده شامل اطلاعات زیر می باشد.

- متغیر setNum : شماره پارامتری که باید ارسال شود.
- متغیر value : مقدار پارامتری که باید ارسال شود.

به عنوان مثال، اگر setNum = 3 و value = 100 باشد، رشته تولید شده به شکل زیر خواهد بود.
Setting P3 = 100

➢ **ارسال داده‌ها از طریق تابع HAL_UART_Transmit_IT**

```
HAL_UART_Transmit_IT(&huart2, (uint8_t*)buffer, strlen(buffer));
```

این خط از تابع HAL_UART_Transmit_IT برای ارسال داده‌های موجود در buffer از طریق UART استفاده می‌کند. پارامترهای ورودی شامل کانال پورت سریال، بافر بسته اطلاعاتی و طول بافر که با استفاده از تابع strlen محاسبه می‌شود.

۶-۴. معرفی و بررسی کتابخانه Motion Profile

۶-۵. معرفی و بررسی کتابخانه Basic Setting

همانطور که قبلا در بخش آموزش رابط کاربری کنترلر توضیح داده بودیم باید از چرخش روتاری انکودر برای افزایش و کاهش اعداد و از دکمه فشاری آن برای جابجا شدن بین ارقام استفاده می کردیم این تابع بدین منظور ایجاد شده است که پارامترهای عددی موجود در تنظیمات را کنترلر برای تنظیم مقادیر صحیح و اعشاری پارامترهای عددی مانند تنظیم سرعت و موقعیت این کتابخانه برای محاسبات مقادیر اعمال شده توسط هر پارامتر تنظیمات مثل پارامتر سرعت ایجاد شد است در این کتابخانه توابعی وجود دارد که با استفاده از آنها می توانید پارامتر های تنظیم عددی هر تنظیمات را انجام دهید.

۶-۶. معرفی و بررسی کتابخانه Advanced Setting

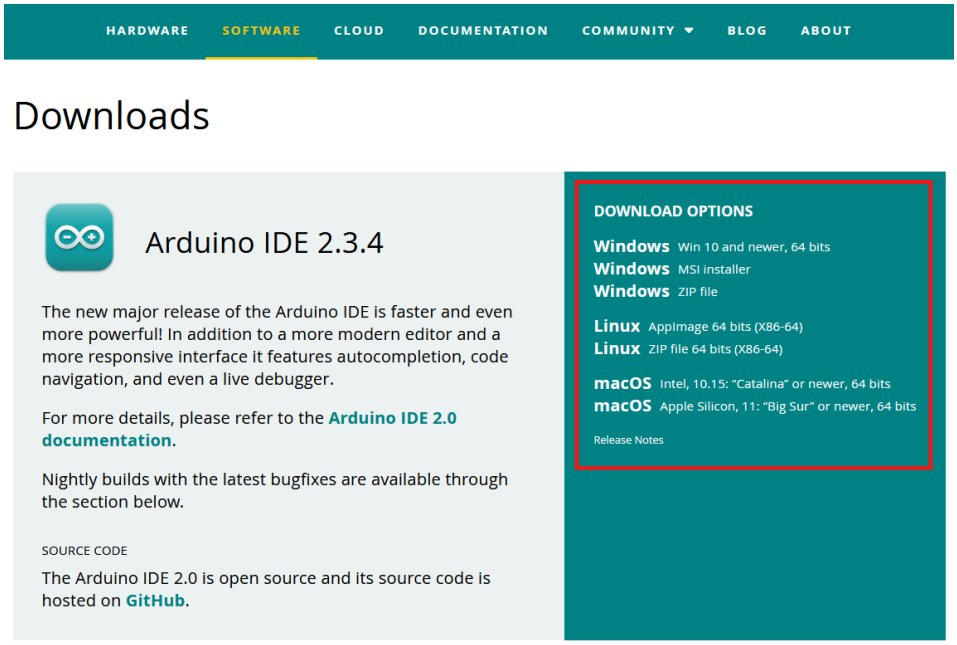
۶-۷. معرفی و بررسی کتابخانه Program Setting

۶-۸. معرفی و بررسی کتابخانه Homing Setting

فصل هفتم: برنامه نویسی وب سرور کنترلر

۷-۱. پیکربندی ماژول ESP8266 در محیط Arduino IDE

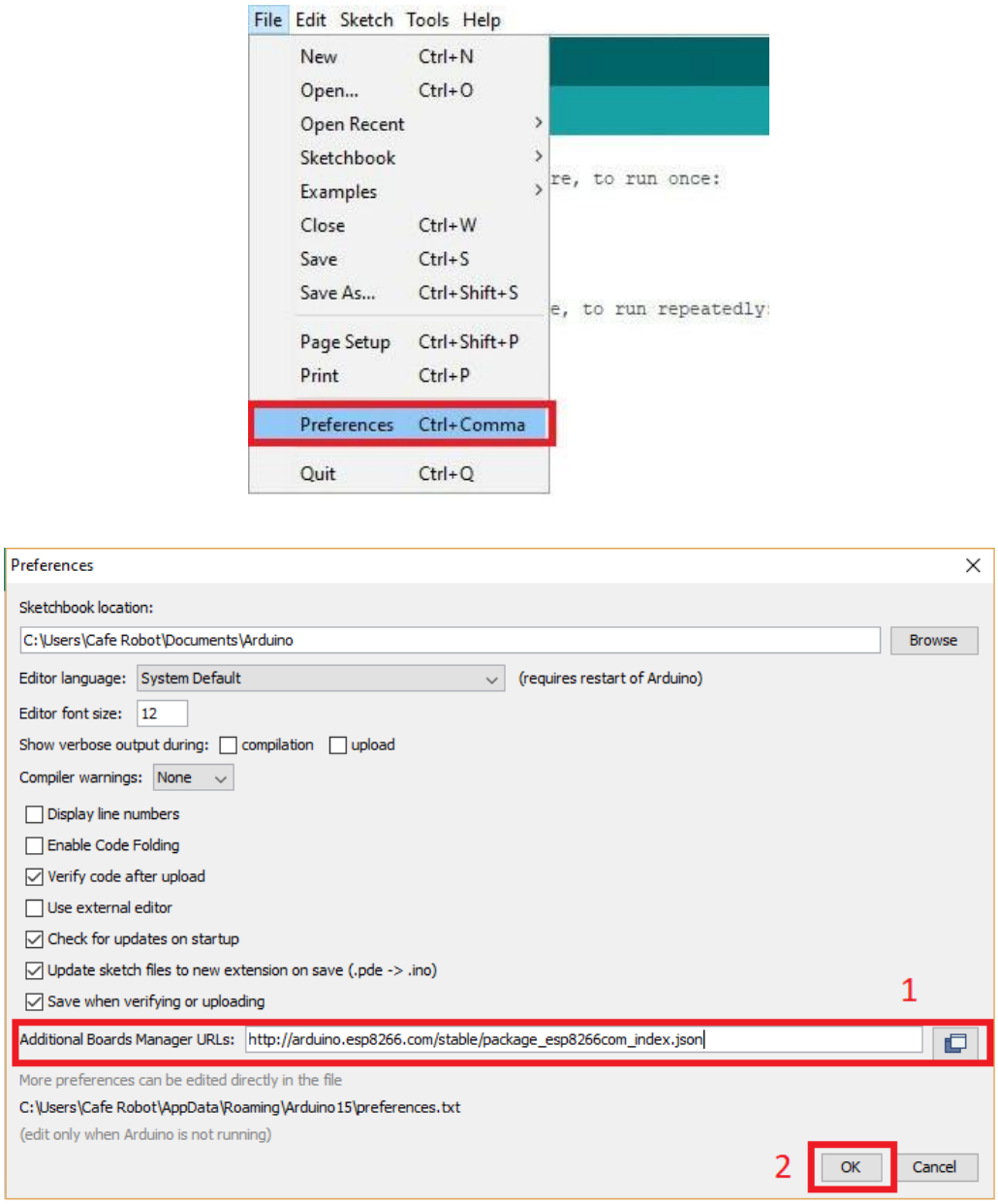
در مرحله اول باید آخرین نسخه Arduino IDE را بر روی کامپیوتر خود نصب کنید. می‌توانید از طریق سایت Arduino.cc ، آخرین نسخه را متناسب با سیستم عامل خود دانلود کنید. کافی است در بخش Download Options ، سیستم عامل خود را انتخاب کنید و بر روی آن کلیک کنید.



شکل ۷- ۱- نصب آخرین نسخه Arduino IDE

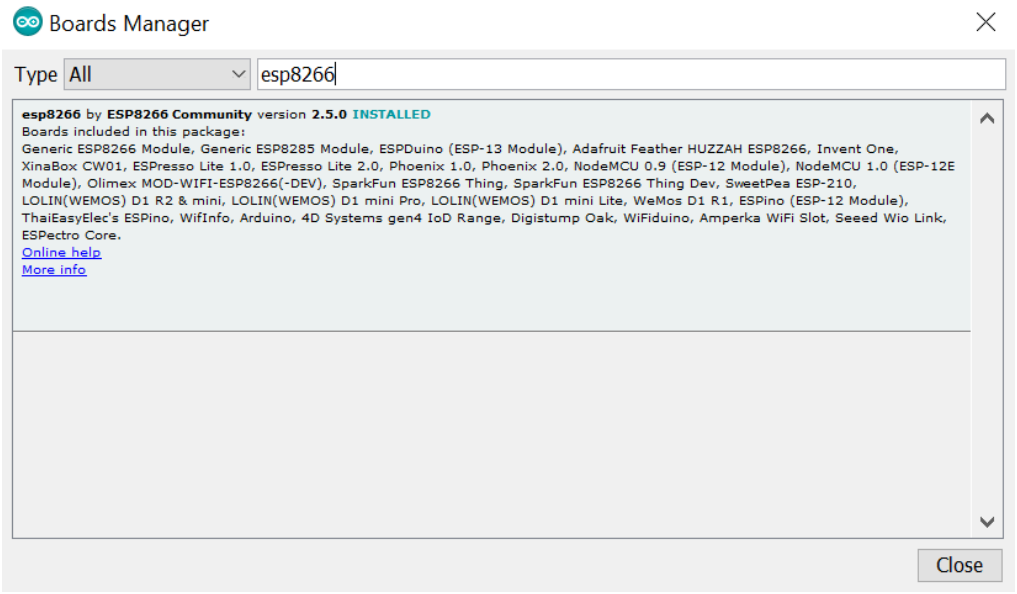
پس از نصب آخرین نسخه آردوینو، برای شروع، باید قسمت board manager را با یک URL خاص آپدیت کنید. نرم افزار آردوینو را باز کرده و مسیر File > Preferences را دنبال کنید. سپس URL زیر را در قسمت Additional Board Manager URLs در پایین صفحه وارد کنید.

http://arduino.esp8266.com/stable/package_esp8266com_index.json



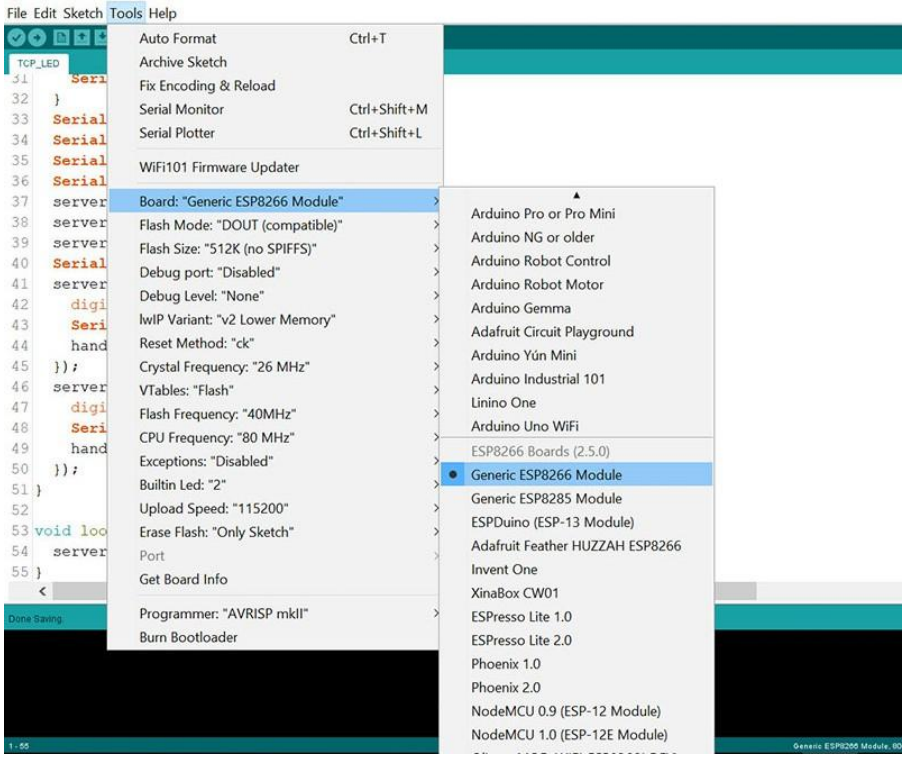
شکل ۷- ۲- جایگذاری URL ماژول ESP8266 در تنظیمات Preferences

سپس، گزینه‌ی OK را انتخاب کنید. پس از آن، از مسیر Tools > Boards > Boards Manager قسمت Boards Manager را باز کنید. اکنون باید گزینه‌های جدیدی به بردهای استاندارد آردوینو اضافه شده باشد. به همین دلیل، با نوشتن عبارت ESP8266 ، جستجوی خود را محدود کنید. در نهایت، بر روی نتیجه‌ی ظاهر شده مانند شکل ۷-۳ کلیک کرده و install را بزنید.



شکل ۷- ۳- نصب کتابخانه ESP8266 در محیط Arduino IDE

زمانی که نصب به پایان رسید می توانید Board Manager را ببندید. سپس برای آپلود کد و اجرای آن باید از روشی که در فصل قبل در بخش معرفی ماژول ESP8266 توضیح داده شد برای پروگرام کردن ماژول استفاده نمایید. اما قبل از آن باید مطمئن شوید که برد ESP8266 در نرم‌افزار آردوینو به درستی انتخاب شده باشد. برای اینکار به مسیر Tools > Board رفته و سپس مدل برد خود را روی Generic ESP8266 Module قرار دهید. (دقت کنید که Upload Speed روی ۱۱۵۲۰۰ باشد)



شکل ۷- ۴- انتخاب نوع ماژول ESP8266 برای آپلود و اجرای کد

۷-۲. معرفی و بررسی کدهای آردوینو

برای شروع برنامه نویسی در محیط آردوینو، اولین قدم شناخت کامل ساختار محیط برنامه نویسی است. ساختار محیط برنامه نویسی آردوینو از دو تابع اصلی تشکیل می شود که با هر بار ورود به نرم افزار آردوینو و ایجاد یک تب جدید برای شروع برنامه نویسی با آنها مواجه خواهید شد.

۷-۲-۱. معرفی تابع `setup`

```
void setup() {
  // put your setup code here, to run once
}
```

این تابع یکی از اساسی ترین بخش های برنامه نویسی در آردوینو است که هنگام راه‌اندازی برد اجرا شده و وظایف مختلفی را برای مقداردهی اولیه سیستم انجام می‌دهد. این وظایف شامل مقداردهی اولیه پورت سریال، حافظه EEPROM ، سیستم فایل، راه‌اندازی وای‌فای و تنظیمات مربوط به سرور وب است.

- **مقداردهی اولیه پورت سریال**

```
Serial.begin(115200);
```

این دستور پورت سریال را با نرخ انتقال داده[1] ۱۱۵۲۰۰ بیت بر ثانیه مقداردهی اولیه می‌کند.

- **مقداردهی اولیه حافظه EEPROM**

```
EEPROM.begin(512);
```

دستور فوق حافظه EEPROM را با اندازه ۵۱۲ بایت مقداردهی اولیه می‌کند. از EEPROM برای ذخیره تنظیمات و داده‌هایی که نیاز است پس از ریست شدن دستگاه باقی بمانند استفاده می‌شود.

- **مقداردهی اولیه سیستم فایل LittleFS**

```
LittleFS.begin();
```

این دستور سیستم فایل LittleFS را مقداردهی اولیه می‌کند. LittleFS ، امکان ذخیره‌سازی فایل‌ها را در حافظه فلش ماژول ESP8266 فراهم می‌کند. از این سیستم فایل می‌توان برای ذخیره تنظیمات، صفحات وب و داده‌های دیگر استفاده کرد.

[1] Baud Rate

راه‌اندازی نقطه دسترسی وای‌فای

```
WiFi.softAP(wifiSSID, wifiPassword);
Serial.println(WiFi.softAPIP());
```

در این بخش ESP8266 به عنوان یک نقطه دسترسی[1] تنظیم می‌شود. متغیرهای `wifiSSID` و `wifiPassword`، نام و رمز عبور شبکه وای‌فای ایجادشده را تعیین می‌کنند. سپس آدرس IP این نقطه دسترسی در پورت سریال نمایش داده می‌شود تا بتوان به آن متصل شد.

مقداردهی اولیه سرور وب و تنظیم مسیرها

```
  webServer.on("/", handleRoot);
  webServer.on("/Send", handleSendDataRequest);
  webServer.on("/Receive", handleReceiveDataRequest);
  webServer.begin();
```

در این بخش، وب‌سرور داخلی روی ESP8266 راه‌اندازی شده و مسیرهای زیر برای آن تعریف می‌شوند.

`"/"` ← نمایش صفحه اصلی وب‌سرور

`"/Send"` ← ارسال داده‌ها به صفحه وب

`"/Receive"` ← دریافت دستورات از صفحه وب

در پایان دستور `webServer.begin()`، وب سرور را راه‌اندازی می‌کند.

مقداردهی اولیه تنظیمات پیش‌فرض و ذخیره در EEPROM

```
int defaultSettings[] = {500,0,2,500,2,500,2,500,2,300,200};
for (int i = 0; i < sizeof(defaultSettings); i++) {
  saveToEEPROM(i, defaultSettings[i]);
}
```

در این بخش، یک آرایه از تنظیمات پیش‌فرض تعریف شده و مقدار هر یک از عناصر آن در حافظه EEPROM ذخیره می‌شود. این کار باعث می‌شود که پس از ریست شدن دستگاه، تنظیمات اولیه بارگذاری شوند.

[1] Access Point

۷-۲-۲. معرفی تابع loop

```
void loop() {
  // put your main code here, to run repeatedly
}
```

این تابع یکی از اساسی ترین بخش های برنامه نویسی در آردوینو است و بدون آن، تداخل در عملکرد میکروکنترلر ایجاد می‌کند. حتی اگر در این قسمت برنامه ای هم نوشته نشده باشد، نیاز به آن خواهد بود. در این تابع برنامه اصلی نوشته می شود و کدهای نوشته شده به صورت نامحدود تکرار می شوند.

- **مدیریت درخواست‌های وب سرور**

```
webServer.handleClient();
```

این خط وظیفه پردازش تمام درخواست‌هایی که از سمت کلاینت (مرورگر یا سایر دستگاه‌ها) ارسال می‌شوند را بر عهده دارد. هر بار که تابع loop اجرا می‌شود، بررسی می‌کند که آیا درخواستی از کلاینت دریافت شده است یا خیر. در صورت دریافت درخواست، آنرا پردازش کرده و پاسخ مناسب را ارسال می‌کند.

- **بررسی ورودی‌های دریافتی از پورت سریال**

```
if (Serial.available() > 0)
```

این شرط بررسی می‌کند که آیا داده‌ای از طریق پورت سریال دریافت شده است یا خیر. اگر مقدار Serial.available() بیشتر از صفر باشد، یعنی داده‌ای از طریق پورت سریال وارد شده است و می‌توان آن را خواند.

- **خواندن داده‌های سریال و ذخیره آن در serialBuffer**

```
serialByte = Serial.read();
serialBuffer += serialByte;
```

دستور Serial.read() ، یک کاراکتر را از ورودی سریال خوانده و در متغیر serialByte ذخیره می‌کند و در خط بعدی به انتهای serialBuffer اضافه می‌کند تا داده‌ها به‌صورت یک رشته کامل ذخیره شوند.

شناسایی انتهای داده دریافتی از سریال و آماده‌سازی برای پردازش

```
if (serialByte == '\n')
```

هنگامی که `serialByte` برابر با `'\n'` (کاراکتر پایان خط) باشد، یعنی یک دستور کامل از طریق سریال دریافت شده است. در این لحظه برنامه آماده پردازش این داده می‌شود.

ذخیره دستور دریافتی و ارسال برای پردازش

```
webCommand = serialBuffer;
processIncomingSerialData(serialBuffer);
```

مقدار `serialBuffer` (که حاوی داده‌ی کامل دریافتی است) در `webCommand` ذخیره می‌شود تا یک نسخه از آن برای صفحه وب ارسال شود. سپس تابع `processIncomingSerialData` فراخوانی می‌شود تا داده‌های دریافتی در این تابع پردازش شده و اقدامات مناسب بر روی تابع های دریافتی انجام شود.

پاک کردن بافر سریال

```
serialBuffer = "";
```

پس از پردازش داده‌ی دریافتی، بافر سریال خالی می‌شود تا برای دریافت داده‌های جدید آماده باشد.

مدیریت حافظه و جلوگیری از سر ریز شدن بافر

```
trimDataBuffer();
```

این تابع اضافه‌بار[1] احتمالی در بافر سریال را کنترل می‌کند. در صورتی که بافر سریال بیش از حد طولانی شود، بخش‌های قدیمی آن حذف می‌شوند تا از اشغال غیر ضروری حافظه جلوگیری شود.

افزودن یک تأخیر کوتاه برای بهبود پایداری برنامه

```
delay(10);
```

این تأخیر کوچک (۱۰ میلی‌ثانیه) باعث کاهش بار پردازشی پردازنده می‌شود. از دریافت داده‌های بیش از حد سریع جلوگیری کرده و به پردازش سریال زمان کافی می‌دهد.

[1] Overflow

۷-۲-۳. معرفی تابع processIncomingSerialData

```
// Process incoming serial data
void processIncomingSerialData(String Data);
```

این تابع وظیفه پردازش داده‌های دریافتی از پورت سریال را دارد و مشخص می‌کند که داده ورودی مربوط به کدام بخش از برنامه است. بسته به نوع داده دریافت‌شده، یکی از توابع پردازشگر مناسب فراخوانی می‌شود.

- **بررسی داده ورودی با آرگومان "Code"**

```
if (Data.indexOf("Code") != -1) {
  handleCode(Data);
}
```

اگر این مقدار در داده ورودی وجود داشته باشد، تابع handleCode اجرا می‌شود.

- **بررسی داده ورودی با آرگومان "Positions"**

```
else if (Data.indexOf("Positions") != -1) {
  handlePositions(Data);
}
```

اگر مقدار "Positions" در داده ورودی یافت شود، تابع handlePositions اجرا می‌شود.

- **بررسی داده ورودی با آرگومان "Setting"**

```
else if (Data.indexOf("Setting") != -1) {
  handleSetting(Data);
}
```

در صورت وجود "Setting" در داده، تابع handleSetting فراخوانی شده و تنظیمات مربوطه اعمال می‌شوند.

➢ **بررسی داده ورودی با آرگومان "Ok"**

```
else if (Data.indexOf("Ok") != -1) {
  processOkCommand();
}
```

در این حالت، تابع processOkCommand اجرا شده و دستور "Ok" پردازش می‌شود.

۷-۲-۴. معرفی تابع processOkCommand

```
// Process the "Ok" command
void processOkCommand();
```

این تابع بطور خاص برای مدیریت پیام‌های "Ok" از سمت کنترلر طراحی شده است. وظیفه آن بررسی وضعیت حرکت سیستم است و اقدام مناسب را بر اساس اینکه حرکت با استفاده از موقعیت‌های ذخیره‌شده یا با استفاده از کدهای حرکتی انجام می‌شود، انجام می‌دهد.

➢ **بررسی حرکت با موقعیت‌های ذخیره‌شده**

```
if (movingByPosition == true)
```

این بخش بررسی می‌کند که آیا سیستم در حال حرکت با استفاده از موقعیت‌های ذخیره‌شده است یا خیر. اگر متغیر movingByPosition مقدار true داشته باشد، یعنی حرکت با موقعیت‌ها درحال انجام است.

➢ **ارسال موقعیت‌ها به کنترلر**

```
if (sendPositionCount < PositionCount) {
  sendPositionCount++;
  sendPositionToController(sendPositionCount);
}
```

در صورتیکه حرکت با موقعیت‌ها فعال باشد، این قسمت از کد بررسی می‌کند که آیا تعداد موقعیت های ارسال‌شده کمتر از تعداد کل موقعیت‌های ذخیره شده (PositionCount) است یا خیر. اگر هنوز موقعیت‌های بیشتری برای ارسال وجود داشته باشد، موقعیت بعدی به کنترلر ارسال می‌شود.

پایان حرکت با مختصات و ارسال پیام پایان

```
else {
  movingByPosition = false;
  sendPositionCount = 1;
  webCommand = "End positions move";
  if (Serial) {
    Serial.println("End");
  }
}
```

اگر تمام موقعیت‌ها ارسال شده باشند، حرکت با موقعیت‌ها به پایان می‌رسد. در این صورت، متغیر `movingByPosition` به `false` تغییر داده می‌شود و شمارنده موقعیت‌های ارسال شده به مقدار اولیه ۱ باز می‌گردد. سپس پیام `"End positions move"` به متغیر `webCommand` اختصاص داده می‌شود تا به صفحه وب ارسال شود و همچنین پیامی از نوع `"End"` به پورت سریال نیز ارسال می‌شود.

بررسی حرکت با کدهای حرکتی[1]

```
if (movingByCode == true)
```

در این بخش بررسی می‌شود که آیا سیستم در حال حرکت با استفاده از دستورات کد است یا خیر. اگر متغیر `movingByCode` مقدار `true` داشته باشد، یعنی سیستم در حال حرکت با کد است.

ارسال خطوط کد به کنترلر

```
if (sendCodeCount < codeLineIndex) {
  sendCodeCount++;
  sendCodeToController(sendCodeCount);
}
```

اگر حرکت با کدهای حرکتی فعال باشد، این قسمت از کد بررسی می‌کند که آیا هنوز خطوط کد بیشتری برای ارسال وجود دارد یا خیر. اگر تعداد خطوط ارسال‌شده کمتر از تعداد کل خطوط کد (`codeLineIndex`) باشد، خط کد بعدی به کنترلر ارسال می‌شود.

[1] **Gcode**

➢ **پایان حرکت با کد و ارسال پیام پایان**

```
else {
  movingByCode = false;
  sendCodeCount = 1;
  webCommand = "End code move";
  if (Serial) {
    Serial.println("End");
  }
}
```

اگر تمام خطوط کد ارسال شده باشند، حرکت با کدهای حرکتی به پایان می‌رسد. در این حالت، متغیر `movingByCode` به `false` تغییر داده می‌شود و شمارنده خطوط کد به مقدار اولیه ۱ باز می‌گردد. همچنین پیام `"End code move"` به متغیر `webCommand` اختصاص داده می‌شود و پیامی از نوع `"End"` به پورت سریال ارسال می‌شود.

۷-۲-۵. معرفی تابع trimDataBuffer

```
// Trim unnecessary data from the input buffer
void trimDataBuffer();
```

این تابع برای محدود کردن طول داده‌های ذخیره‌شده در بافر سریال استفاده می‌شود. در ادامه بخش های مختلف این تابع را بررسی خواهیم کرد.

➢ **بررسی طول بافر سریال**

```
if (serialBuffer.length() > 200)
```

این شرط بررسی می‌کند که آیا طول بافر سریال بیشتر از ۲۰۰ کاراکتر است یا نه. اگر مقدار داده‌ها در این بافر بیش از ۲۰۰ کاراکتر باشد، نیاز به حذف داده‌های اضافی داریم.

➢ **حذف داده‌های اضافی از ابتدای بافر**

```
serialBuffer.remove(0, serialBuffer.length() - 200);
```

این خط از کد، متد `remove` را روی `serialBuffer` فراخوانی می‌کند. این متد دو آرگومان دریافت می‌کند:

آرگومان اول مشخص می‌کند که حذف داده‌ها از ابتدای رشته (`serialBuffer`) آغاز شود. آرگومان دوم مشخص می‌کند که چند کاراکتر باید حذف شود. مقدار آن برابر است با تعداد کاراکترهای اضافی (طول کلی منهای ۲۰۰ کاراکتر آخر). به این ترتیب، اگر بافر سریال مثلاً ۲۵۰ کاراکتر داشته باشد، مقدار ۵۰ از ابتدای رشته حذف می‌شود و فقط ۲۰۰ کاراکتر آخر باقی می‌مانند.

۷-۲-۶. معرفی تابع handleReceiveDataRequest

```
// Handle incoming data request from the web server
void handleReceiveDataRequest();
```

این تابع مسئول پردازش درخواست‌های دریافتی از کلاینت (صفحه وب) است و بر اساس نوع داده ارسال‌شده، اقدامات مختلفی انجام می‌دهد. ماژول ESP8266 می‌تواند داده‌هایی مانند دستورات کنترلی، موقعیت‌ها، تنظیمات و پیام‌های خاص از رابط کاربری دریافت کند و بر اساس آن‌ها پاسخ مناسب به کنترلر و صفحه وب ارسال کند.

➢ تنظیم هدر CORS برای ارتباط از منابع مختلف

```
webServer.sendHeader("Access-Control-Allow-Origin", "*");
```

این خط باعث می‌شود که درخواست‌های ارسال‌شده از دامنه‌های مختلف (CORS) مجاز باشند. این کار مخصوصاً زمانی که کلاینتی از یک دامنه دیگر داده ارسال می‌کند، مفید است.

➢ پردازش درخواست با آرگومان "code"

```
if (webServer.hasArg("code")) {
  String codeCom = webServer.arg("code");
}
```

بررسی می‌کند که آیا درخواست شامل پارامتر `"code"` است یا خیر. درصورت وجود درخواست را دریافت و در متغیر `codeCom` ذخیره می‌کند.

کنترل دستورات مربوط به "code"

```
if (codeCom.indexOf("Pause") != -1) {
  if (Serial) {
    Serial.println("Code Pause");
  }
}
else if (codeCom.indexOf("pBack") != -1) {
  if (Serial) {
    Serial.println("Code pBack");
  }
}
```

بررسی می‌کند که آیا مقدار codeCom شامل پارامتر "Pause" یا "pBack" است:

- اگر مقدار "Pause" باشد، "Code Pause" را در پورت سریال چاپ می‌کند.
- اگر مقدار "pBack" باشد، "Code pBack" را در پورت سریال چاپ می‌کند.

فراخوانی تابع دریافت و ذخیره کدهای حرکتی

```
if (codeCom.indexOf("EndSendingCode") != -1) {
  webCommand = "End";
}
else {
  handleCode(codeCom);
}
```

این شرط کدهای حرکتی دریافتی از صفحه وب را به تابع handleCode هدایت می کند و زمانی که دستور "EndSendingCode" (پایان ارسال کدهای حرکتی) از طرف صفحه وب ارسال شد مقدار webCommand را برابر با "End" می کند یا به عبارت دیگه دستور "End" را به صفحه‌وب ارسال می‌کند. این پیام برای تایید عملکرد ماژول ESP8266 در صفحه وب کاربرد دارد.

پردازش درخواست‌های با آرگومان "Positions"

```
else if (webServer.hasArg("Positions")) {
  String positionCom = webServer.arg("Positions");
}
```

بررسی می‌کند که آیا درخواست شامل پارامتر "Positions" است یا خیر. درصورت وجود درخواست را دریافت و در متغیر positionCom ذخیره می‌کند.

کنترل دستورات مربوط به "Positions"

```
if (positionCom.indexOf("Clear") != -1) {
  if (Serial) {
    Serial.println("Positions Clear");
  }
}
else if (positionCom.indexOf("Pause") != -1) {
  if (Serial) {
    Serial.println("Positions Pause");
  }
}
else if (positionCom.indexOf("pBack") != -1) {
  if (Serial) {
    Serial.println("Positions pBack");
  }
}
```

مشابه بخش code ، بررسی می‌شود که آیا دستورات خاصی (مانند Clear ، Pause ، pBack) در مقدار "Positions" وجود دارد یا خیر. درصورتی هرکدام از این دستورات وجود داشت، دستور مربوطه را برای کنترلر از طریق پورت سریال ارسال می کند.

فراخوانی تابع پردازش موقعیت‌ها

```
handlePositions(positionCom);
```

مقدار دریافت‌شده پردازش شده و در نهایت به تابع handlePositions ارسال می‌شود.

پردازش درخواست با آرگومان "move"

```
else if (webServer.hasArg("move")) {
    String moveCom = webServer.arg("move");
  if (Serial) {
    Serial.println("Move = " + moveCom);
  }
}
```

بررسی می‌کند که آیا درخواست شامل پارامتر "move" است یا خیر. درصورت وجود، داده‌های ارسال شده را دریافت و در متغیر moveCom ذخیره می‌کند. سپس مقدار moveCom را برای کنترلر از طریق سریال ارسال می‌کند.

پردازش درخواست با آرگومان "setting"

```
else if (webServer.hasArg("setting")) {
  String settingCom = webServer.arg("setting");

  handleSetting(settingCom);
  if (Serial) {
    Serial.println(settingCom);
  }
}
```

بررسی می‌کند که آیا درخواست شامل پارامتر "setting" است یا خیر. درصورت وجود، داده‌های دریافتی را داخل متغیر settingCom ذخیره می‌کندو تابع handleSetting را فراخوانی می‌کند. سپس یک نسخه از داده‌های دریافتی را نیز توسط سریال برای کنترلر ارسال می‌کند.

پردازش درخواست با آرگومان "BackEnd"

```
else if (webServer.hasArg("BackEnd")) {
  String backEndCom = webServer.arg("BackEnd");

  if (backEndCom.indexOf("Refresh") != -1) {
    isResendSettingActive = true;
  }
}
```

بررسی می‌کند که آیا درخواست شامل پارامتر "BackEnd" است یا خیر. درصورت وجود، داده‌های دریافتی را در متغیر backEndCom ذخیره می کند. اگر در داده‌های دریافتی پارامتر "Refresh" وجود داشته باشد متغیر isResendSettingActive فعال می‌شود.

مدیریت درخواست‌های نامعتبر

```
else {
  webServer.send(400, "text/plain", "No command received.");
}
```

در طول اجرای تابع، ابتدا بررسی می‌شود که آیا درخواست دریافتی شامل یکی از پارامتر های "code"، "Positions"، "move"، "setting" یا "BackEnd" هست یا خیر. اگر هیچ یک از این پارامترها در درخواست موجود نباشد، برنامه وارد این بخش می‌شود.

در این بخش، سرور یک پاسخ HTTP با کد خطای ۴۰۰ و متن `"No command received."` به کلاینت ارسال می‌کند. این مقدار نشان می‌دهد که درخواست کلاینت نامعتبر بوده است و شامل هیچ یک از پارامترهای شناخته‌شده برای سرور نیست.

۷-۲-۷. معرفی تابع handleSendDataRequest

```
// Handle sending data to the web server
void handleSendDataRequest();
```

این تابع مسئول ارسال داده‌ها از سرور (ماژول ESP8266) به کلاینت (صفحه وب) است. داده‌های ارسالی شامل اطلاعاتی هستند که باید در پاسخ به درخواست‌های کلاینت ارسال شوند.

- **ارسال هدر برای اجازه دسترسی از منابع دیگر (CORS)**

```
webServer.sendHeader("Access-Control-Allow-Origin", "*");
```

این خط هدر CORS را تنظیم می‌کند تا اجازه دسترسی به سرور از هر دامنه‌ای (`"*"`) را بدهد. این کار معمولاً برای ارتباط بین کلاینت‌های مختلف (مثلا مرورگر و سرور) انجام می‌شود.

- **بررسی نیاز صفحه وب به بازیابی یا ارسال مجدد تنظیمات یا موقعیت های ذخیره شده**

```
if (isResendSettingActive) {
  webCommand = getResentSettingData();
}
else if (isResendPositionActive) {
  webCommand = getResentPositionsData();
}
```

این بخش بررسی می‌کند که آیا باید داده‌های تنظیمات یا موقعیت‌های ذخیره شده در ماژول ESP8266 را به صفحه وب ارسال کند یا خیر. درصورت تازه سازی[1] صفحه وب عمل بازیابی تنظیمات و موقعیت ها مورد نیاز است.

اگر `isResendSettingActive` فعال باشد (صفحه وب نیاز به بازیابی تنظیمات داشته باشد)، مقدار `webCommand` برابر با خروجی تابع `getResentSettingData` می شود تا تنظیمات به صفحه وب ارسال شود.

[1] Refresh

اگر isResendPositionActive فعال باشد (صفحه وب نیاز به بازیابی موقعیت ها داشته باشد) مقدار webCommand برابر با خروجی تابع getResentPositionsData می شود تا موقعیت ها به صفحه‌وب ارسال شود.

➢ ارسال پاسخ به صفحه‌وب و بررسی عدم تکرار در ارسال

```
if (lastSentData != webCommand || webCommand == "End code move" ||
webCommand == "End positions move" || webCommand == "End") {
  lastSentData = webCommand;
  webServer.send(200, "text/plain", webCommand);
  webCommand = " ";
}
else {
  webServer.send(200, "text/plain", " ");
}
```

اگر مقدار جدید webCommand با مقدار آخرین داده ارسال‌شده (lastSentData) متفاوت باشد یا مقدار خاصی مثل "End code move" یا "End positions move" یا "End" باشد:

مقدار lastSentData برابر با webCommand قرار می‌گیرد. (برای بررسی عدم تکرار در ارسال داده) و مقدار webCommand به کلاینت (صفحه وب) ارسال می‌شود. سپس مقدار webCommand پاک می‌شود. اگر هیچ تغییری در داده‌ها رخ نداده باشد رشته خالی به صفحه وب ارسال می‌شود.

۷-۲-۸. معرفی تابع handleCode

```
// Process the Code command
void handleCode(String codeData);
```

این تابع مسئول پردازش دستورات حرکتی دریافتی از سمت کلاینت (صفحه وب) یا کنترلر است.

➢ بررسی دستور "Run" (اجرای حرکت)

```
if (codeData.indexOf("Run") != -1) {
  if (codeLineIndex > 0 && movingByCode != true) {
    movingByCode = true;
    sendCodeToController(sendCodeCount);
  }
}
```

اگر دستور `"Run"` در `codeData` وجود داشته باشد، همچنین اگر داده‌ای (کد حرکتی) برای اجرا وجود داشته باشد (`codeLineIndex > 0`) و آیا اگر در حال حاضر حرکتی در حال اجرا نباشد (`movingByCode != true`) در این صورت، حالت حرکت فعال می‌شود (`movingByCode = true`) و اولین دستور حرکتی به کنترلر ارسال می‌شود. (تابع `sendCodeToController` فراخوانی می‌شود)

- **بررسی دستور `"Pause"` (مکث یا توقف حرکت)**

```
else if (codeData.indexOf("Pause") != -1) {
  movingByCode = false;
}
```

اگر دستور `"Pause"` در `codeData` وجود داشته باشد، متغیر `movingByCode` مقدار `false` می‌گیرد. این باعث می‌شود حرکت متوقف شود بدون اینکه داده‌های حرکتی فراموش یا پاک شوند.

- **بررسی دستور `"pBack"`** (بازگشت به اولین گام)

```
else if (codeData.indexOf("pBack") != -1) {
  movingByCode = false;
  sendCodeCount = 1;
}
```

اگر دستور `"pBack"` در `codeData` وجود داشته باشد، متغیر `movingByCode` غیرفعال شده و مقدار `sendCodeCount` برابر ۱ می‌شود. بدین ترتیب حرکت متوقف شده و برنامه حرکت از سر گرفته می‌شود.

- **بررسی دستور `"Send"` (ارسال داده‌های حرکتی به ماژول ESP8266)**

```
else if (codeData.indexOf("Send") != -1) {
  webCommand = "Send";
}
```

اگر دستور `"Send"` در `codeData` وجود داشته باشد، مقدار `webCommand` برابر `"Send"` قرار می‌گیرد تا به صفحه وب اطلاع داده شود که ارسال داده‌ها در حال انجام است.

بررسی دستور "Clear" (پاک کردن داده‌های حرکتی)

```
else if (codeData.indexOf("Clear") != -1) {
  codeLineIndex = 0;
  sendCodeCount = 1;
  movingByCode = false;
  for (int i = 0; i <= 1500; i++) {
    CodeX[i] = NAN;
    CodeY[i] = NAN;
    CodeZ[i] = NAN;
  }
}
```

اگر دستور **"Clear"** در codeData وجود داشته باشد، مقدار codeLineIndex صفر شده و شمارنده sendCodeCount روی مقدار اولیه (۱) تنظیم می‌شود. اگر حرکت با کدها فعال باشد آن را غیرفعال می‌کند. (movingByCode = false) همچنین آرایه‌های CodeX ، CodeY و CodeZ پاک می‌شوند و مقدار NAN می‌گیرند. این دستور برای پاک سازی کامل داده‌های حرکتی ذخیره شده استفاده می‌شود.

پردازش داده‌های حرکتی

```
else {
  codeLineIndex = extractValueFromCode(codeData, "(");
  CodeX[codeLineIndex] = extractValueFromCode(codeData, "X");
  CodeY[codeLineIndex] = extractValueFromCode(codeData, "Y");
  CodeZ[codeLineIndex] = extractValueFromCode(codeData, "Z");
}
```

اگر codeData هیچ‌کدام از دستورات بالا را شامل نشود، فرض می‌شود که یک دستور حرکتی جدید دریافت شده است. مقدار codeLineIndex از کد استخراج شده و مختصات X ، Y و Z در آرایه‌های مربوطه ذخیره می‌شوند. این مختصات بعداً برای اجرای حرکت پردازش خواهند شد.

۷-۲-۹. معرفی تابع handlePositions

```
// Process the Position command (includes movement and delays)
void handlePositions(String PositionData);
```

این تابع وظیفه پردازش و مدیریت داده‌های موقعیتی دریافت‌شده از صفحه وب و کنترلر را بر عهده دارد. این تابع بررسی می‌کند که آیا دستور خاصی (مانند اجرای حرکت، توقف، پاک‌سازی داده‌ها و غیره) در داده‌های دریافت‌شده وجود دارد و سپس اقدامات لازم را انجام می‌دهد.

- **بررسی دستور `Run` (اجرای حرکت)**

```
if (PositionData.indexOf("Run") != -1) {
  if (PositionCount > 0 && movingByPosition != true) {
    movingByPosition = true;
    sendPositionToController(sendPositionCount);
  }
}
```

اگر داده‌های ورودی (PositionData) شامل دستور `"Run"` باشد، موقعیتی برای اجرا موجود باشد (`PositionCount>0`) و فرآیند حرکت موقعیت‌ها در حال اجرا نباشد، (`movingByPosition != true`) مقدار `movingByPosition` به `true` تغییر می‌یابد (فعال‌سازی حرکت بر اساس موقعیت‌ها) سپس اولین موقعیت برای اجرا به کنترلر ارسال می‌شود. این مکانیزم اجازه می‌دهد که حرکت از اولین موقعیت ثبت‌شده آغاز گردد، مشروط بر آنکه موقعیتی برای اجرا موجود باشد.

- **بررسی و اجرای دستور `"Pause"` (مکث یا توقف حرکت)**

```
else if (PositionData.indexOf("Pause") != -1) {
  movingByPosition = false;
}
```

اگر دستور `"Pause"` در داده‌های دریافتی موجود باشد، مقدار `movingByPosition` برابر `false` قرار می‌گیرد. این عملیات باعث توقف حرکت جاری می‌شود، اما داده‌های مربوط به موقعیت‌های ذخیره‌شده پاک نخواهند شد. به محض دریافت دستور `"Run"` مجدداً اجرای حرکت از موقعیت فعلی ادامه خواهد یافت.

بررسی و اجرای دستور "pBack" (بازگشت به اولین موقعیت)

```
else if (PositionData.indexOf("pBack") != -1) {
  movingByPosition = false;
  sendPositionCount = 1;
}
```

اگر دستور "pBack" در داده‌های دریافتی موجود باشد، مقدار movingByPosition برابر false شده و مقدار sendPositionCount به ۱ تغییر می‌یابد. این فرمان حرکت را متوقف کرده و شمارنده موقعیت را به مقدار اولیه تنظیم می‌کند. در نتیجه در اجرای بعدی، سیستم حرکت را از اولین موقعیت ذخیره‌شده مجدداً آغاز خواهد کرد.

بررسی و اجرای دستور "Clear" (پاک‌سازی داده‌های موقعیت)

```
else if (PositionData.indexOf("Clear") != -1) {
  PositionCount = 0;
  sendPositionCount = 1;
  movingByPosition = false;
  for (int i = 0; i < 100; i++) {
    PositionsX[i] = NAN;
    PositionsY[i] = NAN;
    PositionsZ[i] = NAN;
    PositionsDelay[i] = NAN;
  }
}
```

اگر دستور "Clear" در داده‌های دریافتی موجود باشد، مقادیر زیر تنظیم مجدد می‌شوند:

- PositionCount = 0 تعداد موقعیت‌های ذخیره‌شده صفر می شود.
- sendPositionCount = 1 شمارنده ارسال موقعیت‌ها به مقدار اولیه تنظیم می‌شود.
- movingByPosition = false حرکت بر اساس موقعیت ها متوقف می‌شود.
- یک حلقه تمام مقادیر مربوط به مختصات موقعیت‌های ذخیره‌شده را پاک می‌کند.

➢ **پردازش و ذخیره موقعیت‌های جدید**

```
else {
  PositionCount = extractValueFromCode(PositionData, "I");
  PositionsX[PositionCount] = extractValueFromCode(PositionData,
"X");
  PositionsY[PositionCount] = extractValueFromCode(PositionData,
"Y");
  PositionsZ[PositionCount] = extractValueFromCode(PositionData,
"Z");
  PositionsDelay[PositionCount] =
extractValueFromCode(PositionData, "T");
  if (Serial) {
    Serial.println("++");
  }
}
```

اگر ورودی دریافتی هیچ‌کدام از دستورات قبلی (Run ، Pause ، pBack و غیره) را شامل نشود، سیستم فرض می‌کند که داده‌های موقعیتی جدید دریافت شده‌اند. مقدار PositionCount از داده‌های ورودی استخراج شده و مختصات X ، Y و Z همراه با مقدار زمان تأخیر (T) در آرایه‌های مربوطه ذخیره می‌شود. در نهایت، یک پیام تأیید "++" از طریق پورت سریال ارسال می‌شود که نشان می‌دهد داده‌های جدید با موفقیت ذخیره شده‌اند. این بخش مسئول دریافت و ثبت موقعیت‌های جدید جهت اجرا در آینده است. همچنین با استفاده از تابع extractValueFromCode ، مختصات و اطلاعات اضافی از داده‌های ورودی استخراج شده و در آرایه‌های مناسب ذخیره می‌شوند.

۷-۲-۱۰. معرفی تابع handleSetting

```
// Process the Setting command
void handleSetting(String settingData);
```

این تابع وظیفه پردازش داده‌های تنظیماتی دریافت‌شده را بر عهده دارد و این داده‌ها را در حافظه EEPROM ذخیره می‌کند.

➢ **تعریف متغیرهای محلی**

```
int address;
int value;
```

دو متغیر address و value از نوع int تعریف شده‌اند. متغیر address نشان‌دهنده آدرس حافظه EEPROM است که مقدار تنظیمی در آن ذخیره خواهد شد و متغیر value مقدار تنظیماتی است که از داده‌های ورودی استخراج می‌شود و باید در EEPROM ذخیره گردد.

- **استخراج آدرس و مقدار از settingData**

```
address = extractValueFromCode(settingData, "P");
value = extractValueFromCode(settingData, "=");
```

از داده‌های تنظیماتی (settingData) مقدار مربوط به پارامتر "P" استخراج می‌شود و در متغیر address ذخیره می‌گردد. مقدار "P" معمولاً نشان‌دهنده آدرس پارامتر تنظیمات است که باید مقدار تنظیمی با این آدرس داخل حافظه EEPROM ذخیره شود. مقدار موجود بعد از علامت تساوی از داده‌های ورودی استخراج و در متغیر value ذخیره می‌شود. این مقدار نشان‌دهنده مقدار تنظیمی جدید است که باید در سیستم اعمال با ادرس تایین شده در بالا ذخیره گردد.

- **ذخیره مقدار در EEPROM**

```
saveToEEPROM(address, value);
```

تابع saveToEEPROM مسئول ذخیره داده‌ها در حافظه EEPROM مطابق با آدرس و مقدار تنظیمی که به آن ارسال می شود است.

۷-۲-۱۱. معرفی تابع sendPositionToController

```
// Send a Position to the Controller
void sendPositionToController(int lineIndex);
```

این تابع برای ارسال موقعیت‌های ذخیره‌شده به کنترلر استفاده می‌شود.

- **تعریف بافر برای ارسال داده‌ها**

```
char sendBuffer[50];
```

یک آرایه از نوع char به نام sendBuffer با ظرفیت ۵۰ کاراکتر تعریف شده است. این بافر برای فرمت بندی و ذخیره داده‌های موقعیت قبل از ارسال به کنترلر استفاده می‌شود.

➢ **اعمال تأخیر قبل از ارسال موقعیت (در صورت نیاز)**

```
if (lineIndex > 0 && PositionsDelay[lineIndex] > 0) {
  delay(PositionsDelay[lineIndex - 1]);
}
```

اگر مقدار lineIndex بزرگ‌تر از صفر باشد (یعنی حداقل یک موقعیت معتبر وجود داشته باشد) همچنین اگر مقدار تأخیر PositionsDelay[lineIndex] بزرگ‌تر از صفر باشد. (برای موقعیت مورد نظر تاخیر ثبت شده باشد)، تابع delay مقدار تأخیر مشخص‌شده برای هر موقعیت را قبل از ارسال موقعیت بعدی به کنترلر، اعمال می‌کند.

➢ **فرمت‌بندی داده‌های موقعیت و ذخیره در sendBuffer**

```
sprintf(sendBuffer, "Position I%d = X%.2f Y%.2f Z%.2f", lineIndex,
PositionsX[lineIndex], PositionsY[lineIndex],
PositionsZ[lineIndex]);
```

مقادیر موقعیت مورد نظر برای ارسال در قالب یک رشته متنی توسط (sprintf) فرمت‌بندی‌شده و در متغیر sendBuffer ذخیره می‌شوند.

➢ **ارسال داده‌های موقعیت از طریق پورت سریال**

```
if (Serial) {
  Serial.println(sendBuffer);
}
```

در صورت فعال بودن پورت سریال، مقدار ذخیره‌شده در sendBuffer به پورت سریال ارسال می‌شود.

۷-۲-۱۲. معرفی تابع sendCodeToController

```
// Send a Code line to the Controller
void sendCodeToController(int lineIndex);
```

این تابع وظیفه ارسال کدهای حرکتی به کنترلر اصلی را بر عهده دارد. این تابع از کدهای حرکتی ذخیره‌شده در آرایه‌های CodeX ، CodeY و CodeZ استفاده کرده و در قالب یک رشته فرمت بندی شده، اطلاعات را به پورت سریال ارسال می‌کند.

تعریف بافر برای ذخیره داده‌ها

```
char sendBuffer[100];
char tempBuffer[20];
```

sendBuffer یک آرایه ۱۰۰ بایتی است که رشته نهایی ارسالی را در خود ذخیره می‌کند. tempBuffer یک آرایه موقت ۲۰ بایتی است که برای ذخیره هر بخش از داده‌ها قبل از اضافه شدن به sendBuffer استفاده می‌شود.

نوشتن شناسه خط (Index) در sendBuffer

```
int offset = snprintf(sendBuffer, sizeof(sendBuffer), "Code I%d=",
lineIndex);
```

"Code I%d=" مقدار lineIndex که شماره خط دستورات حرکتی را نشان می‌دهد، به‌صورت در sendBuffer ذخیره می‌شود. همچنین متغیر offset تعریف شده و تعداد کاراکترهایی که در sendBuffer ذخیره شده‌اند در آن قرار می‌گیرد.

افزودن مقدار X به sendBuffer

```
if (!isnan(CodeX[lineIndex])) {
  offset += snprintf(tempBuffer, sizeof(tempBuffer), " X%.2f",
CodeX[lineIndex]);
  strncat(sendBuffer, tempBuffer, sizeof(sendBuffer) - offset -
1);
}
```

ابتدا بررسی می‌شود که مقدار CodeX[lineIndex] نامعتبر (NaN) نباشد، یعنی دارای مقدار باشد. اگر شرط برقرار بود مقدار X با دو رقم اعشار در tempBuffer ذخیره می‌شود. سپس تعداد کاراکتر های ایجاد شده در tempBuffer نیز به متغیر offset اضافه می‌شود. سپس مقدار tempBuffer به sendBuffer اضافه می‌شود.

➢ افزودن مقدار Y به sendBuffer

```
if (!isnan(CodeY[lineIndex])) {
  offset += snprintf(tempBuffer, sizeof(tempBuffer), " Y%.2f",
CodeY[lineIndex]);
  strncat(sendBuffer, tempBuffer, sizeof(sendBuffer) - offset -
1);
}
```

مانند قسمت قبل ابتدا بررسی می‌شود که مقدار CodeY[lineIndex] نامعتبر (NaN) نباشد، یعنی دارای مقدار باشد. اگر شرط برقرار بود مقدار Y با دو رقم اعشار در tempBuffer ذخیره می‌شود. سپس تعداد کاراکتر های ایجاد شده در tempBuffer نیز به متغیر offset اضافه می‌شود. سپس مقدار tempBuffer به sendBuffer اضافه می‌شود.

➢ افزودن مقدار Z به sendBuffer

```
if (!isnan(CodeZ[lineIndex])) {
  snprintf(tempBuffer, sizeof(tempBuffer), " Z%.2f",
CodeZ[lineIndex]);
  strncat(sendBuffer, tempBuffer, sizeof(sendBuffer) - offset -
1);
}
```

ابتدا بررسی می‌شود که مقدار CodeZ[lineIndex] نامعتبر (NaN) نباشد یعنی دارای مقدار باشد . اگر شرط برقرار بود مقدار Z با دو رقم اعشار در tempBuffer ذخیره می‌شود . سپس تعداد کاراکتر های ایجاد شده در tempBuffer نیز به متغیر offset اضافه می‌شود . سپس مقدار tempBuffer به sendBuffer اضافه می‌شود

➢ ارسال فرم آماده شده حرکتی از طریق پورت سریال برای کنترلر

```
if (Serial) {
  Serial.println(sendBuffer);
}
```

اگر پورت سریال فعال باشد، مقدار نهایی sendBuffer برای کنترلر از طریق ارتباط سریال ارسال می شود.

۷-۲-۱۳. معرفی تابع getResentSettingData

```
// Send data back to the web
String getResentSettingData();
```

این تابع وظیفه خواندن مقادیر تنظیمات ذخیره‌شده در EEPROM و ارسال آن‌ها به صفحه وب به صورت رشته فرمت بندی شده را بر عهده دارد. این تابع معمولا در راه اندازی و یا بازیابی صفحه‌وب کاربرد دارد.

- **تعریف بافر برای ذخیره داده‌های تنظیمات**

```
char sendBuffer[40];
```

sendBuffer یک آرایه ۴۰ بایتی از نوع char است که برای نگهداری رشته نهایی داده قبل از ارسال استفاده می‌شود. این بافر حداکثر ۴۰ کاراکتر را می‌تواند در خود نگه دارد.

- **خواندن تنظیمات از EEPROM و قرار دادن آن در خروجی تابع**

```
if (settingResendIndex < 11) {
  settingResendIndex++;
  sprintf(sendBuffer, "Setting P%d =%d \n", (settingResendIndex -
1), readFromEEPROM(settingResendIndex - 1));
  return sendBuffer;
}
```

بررسی می‌کند اگر مقدار settingResendIndex کوچک‌تر از ۱۱ باشد. (یعنی هر ۱۰ پارامتر تنظیمات ارسال شده باشند) مقدار settingResendIndex را یک واحد افزایش می‌دهد. همچنین مقدار ذخیره‌شده در EEPROM برای آدرس settingResendIndex - 1 خوانده شده و در قالب یک رشته فرمت بندی می‌شود. رشته تنظیمات ساخته‌شده در sendBuffer را به عنوان خروجی تابع بر می‌گرداند.

- **بررسی اتمام ارسال تنظیمات**

```
else {
  settingResendIndex = 0;
  isResendSettingActive = false;
  isResendPositionActive = true;
  return " ";
```

```
}
```

وقتی تمام تنظیمات ارسال شد (مقدار settingResendIndex به ۱۱ برسد):

- مقدار settingResendIndex ریست می‌شود تا در آینده مجدداً از ابتدا تنظیمات ارسال شوند.
- مقدار isResendSettingActive غیرفعال (false) می‌شود، به این معنی که فرایند ارسال مجدد تنظیمات به صفحه‌وب به پایان رسیده است.
- مقدار isResendPositionActive فعال (true) می‌شود، به این معنی که پس از تنظیمات، ارسال مجدد موقعیت‌ها شروع شود.
- یک رشته خالی برمی‌گرداند که نشان‌دهنده پایان ارسال داده‌های تنظیمات است.

۷-۲-۱۴. معرفی تابع getResentPositionsData

```
// Send data back to the web
String getResentPositionsData();
```

این تابع وظیفه ارسال مجدد موقعیت‌های ذخیره‌شده به صفحه وب را بر عهده دارد. این تابع معمولاً در فرآیند بازیابی یا راه اندازی مجدد صفحه‌وب کاربرد دارد.

➢ تعریف یک بافر برای ذخیره داده‌ها

```
char buffer[50];
```

یک آرایه ۵۰ بایتی از نوع char با نام buffer تعریف شده است که برای ذخیره اطلاعات موقعیت‌های حرکتی قبل از ارسال استفاده می‌شود.

➢ بررسی مقدار PositionResendIndex و ارسال اطلاعات موقعیت

```
if (PositionResendIndex < PositionCount) {
  PositionResendIndex++;
  sprintf(buffer, "Positions I%d = X%.2f Y%.2f Z%.2f T%.2f\n",
PositionResendIndex, PositionsX[PositionResendIndex],
PositionsY[PositionResendIndex], PositionsZ[PositionResendIndex],
PositionsDelay[PositionResendIndex]);
  return buffer;
}
```

بررسی می‌کند اگر مقدار PositionResendIndex کوچکتر از PositionCount باشد یعنی هنوز موقعیت‌هایی برای ارسال وجود داشته باشد، ابتدا مقدار PositionResendIndex را یکی افزایش می‌دهد (به موقعیت بعدی می‌رود). سپس رشته‌ای را با قالب مشخص شامل شماره موقعیت، مقادیر X ، Y ، Z و تأخیر T برای هر موقعیت تولید می‌کند. این رشته را به عنوان خروجی تابع با زمی‌گرداند.

➢ بررسی اتمام ارسال موقعیت‌ها

```
else {
  PositionResendIndex = 0;
  isResendPositionActive = false;
  return "Start";
}
```

اگر تمام موقعیت‌ها ارسال شده باشند:

- مقدار PositionResendIndex ریست می‌شود تا در آینده دوباره از ابتدا ارسال انجام شود .
- مقدار isResendPositionActive غیرفعال (false) می‌شود، یعنی ارسال موقعیت‌ها متوقف می‌شود.
- مقدار "Start" را بر می‌گرداند که نشان‌دهنده شروع عملکرد صفحه وب می‌باشد.

۷-۲-۱۵. معرفی تابع extractValueFromCode

```
// Extract a numerical value from a Code & Positions string based
on the given parameter
float extractValueFromCode(String Data, String parameter);
```

این تابع برای استخراج مقدار عددی یک پارامتر مشخص از یک رشته[1] طراحی شده است. این تابع معمولاً در استخراج مقادیر موقعیت‌ها و کد های حرکتی کاربرد دارد .

➢ دریافت ورودی‌ها

این تابع دو ورودی دریافت می‌کند:

Data ← رشته‌ای که شامل داده‌های متنی دستور است. (مثلاً X12.5 Y34.7 Z8.9)

parameter ← پارامتری که مقدار آن باید استخراج شود. (مثلاً X ، Y ، Z)

[1] String

جستجوی محل قرارگیری پارامتر در رشته

```
int index = Data.indexOf(parameter);
```

از دستور indexOf استفاده شده است تا محل اولین وقوع parameter در رشته Data پیدا شود. اگر parameter در Data وجود داشته باشد، index مکان اولین کاراکتر پارامتر را ذخیره می‌کند.

استخراج مقدار عددی بعد از پارامتر

```
if (index != -1) {
  return Data.substring(index + 1).toFloat();
}
```

بررسی می‌شود اگر مقدار index معتبر باشد (!= -1)، یعنی parameter در Data یافت شده باشد. ابتدا Data.substring(index + 1) بخشی از رشته را که شامل مقدار عددی بعد از پارامتر است جدا می‌کند. سپس مقدار جدا شده توسط دستور toFloat به عدد float تبدیل شده و باز گردانده می‌شود.

اگر پارامتر مورد نظر در داده وجود نداشته باشد

```
return NAN;
```

اگر مقدار index == -1 باشد (یعنی parameter در Data وجود نداشته باشد)، تابع مقدار NAN را باز می‌گرداند. مقدار NAN (Not A Number) نشان دهنده این است که مقدار مورد نظر یافت نشده یا نامعتبر است.

۷-۲-۱۶. معرفی تابع saveToEEPROM

```
// Save a value to EEPROM
void saveToEEPROM(int address, int value);
```

این تابع برای ذخیره مقدار تنظیمات (value) در آدرس مشخصی از حافظه EEPROM طراحی شده است. این تابع ابتدا مقدار موجود در EEPROM را بررسی کرده و در صورتی که مقدار قبلاً در همان آدرس ذخیره شده باشد، از نوشتن مجدد جلوگیری می‌کند. در غیر این صورت، مقدار جدید را ذخیره می‌کند.

بررسی مقدار ذخیره‌شده در EEPROM

```
if (readFromEEPROM(address) == value) {
  return;
}
```

دستور readFromEEPROM مقدار ذخیره شده در آدرس مشخص‌شده را خوانده و با مقدار جدید (value) مقایسه می‌کند. اگر مقدار جدید برابر مقدار فعلی در EEPROM باشد، تابع بلافاصله تمام می‌شود (return) و از نوشتن مجدد داده‌ها جلوگیری می‌شود.

محاسبه آدرس نهایی در EEPROM

```
address = address * 2;
```

هر مقدار int در EEPROM دو بایت فضا اشغال می‌کند. برای جلوگیری از هم‌پوشانی داده‌ها، هر آدرس در EEPROM در ۲ ضرب می‌شود تا مکان درستی برای ذخیره‌سازی مقدار int مشخص شود.

نوشتن مقدار در حافظه EEPROM

```
EEPROM.write(address, value & 0xFF);              // LSB
EEPROM.write(address + 1, (value >> 8) & 0xFF);   // MSB
```

حافظه EEPROM در ماژول ESP8266 به صورت بایت به بایت عمل می‌کند. چون int دارای ۱۶ بیت است، باید در دو آدرس جداگانه ذخیره شود.

بایت کم ارزش (LSB) : value & 0xFF فقط ۸ بیت پایینی (LSB) را استخراج و ذخیره می‌کند.

بایت پر ارزش (MSB) : (value >> 8) & 0xFF مقدار value ، ۸ بیت به راست شیفت داده شده تا فقط ۸ بیت بالایی (MSB) باقی بماند و ذخیره شود.

جدول ۶- ۱- مقدار value و نحوه ذخیره سازی

مقدار value (int)	مقدار LSB (8 بیت پایین)	مقدار MSB (8 بیت بالا)
1025 (0x0401)	0x01 (1)	0x04 (4)
258 (0x0102)	0x02 (2)	0x01 (1)

ذخیره‌سازی نهایی در EEPROM

```
EEPROM.commit();
```

در برخی از بردها مانند ESP8266 و ESP32 ، نوشتن در EEPROM بلافاصله ذخیره نمی‌شود و نیاز به فراخوانی دستور `EEPROM.commit()` دارد تا تغییرات در حافظه دائمی ثبت شوند.

۷-۲-۱۷. معرفی تابع readFromEEPROM

```
// Read a value from EEPROM
int readFromEEPROM(int address);
```

این تابع برای خواندن یک مقدار صحیح (`int`) ذخیره‌شده در حافظه EEPROM استفاده می‌شود.

محاسبه آدرس نهایی در EEPROM

```
address = address * 2;
```

همانطور که می دانید هر مقدار `int` ، دو بایت فضا اشغال می‌کند. از آنجایی که EEPROM داده‌ها را به صورت بایت به بایت ذخیره می‌کند، آدرس هر `int` باید در دو آدرس متوالی قرار گیرد. برای دسترسی به مقدار صحیح ذخیره‌شده، آدرس اولیه در ۲ ضرب می‌شود تا مکان دقیق داده مشخص شود.

جدول ۶- ۲- نحوه تغییر آدرس

مقدار address	مقدار نهایی address × 2
0	0
1	2
2	4

خواندن دو بایت از EEPROM

```
int lowByte = EEPROM.read(address);       // LSB
int highByte = EEPROM.read(address + 1);  // MSB
```

دستور `EEPROM.read(address)` مقدار ذخیره‌شده در آدرس مشخص‌شده را می‌خواند. مقدار `int` در دو آدرس ذخیره شده است، بنابراین بایت اول (LSB - کم‌ارزش‌ترین بایت) از آدرس اولیه خوانده می‌شود. بایت دوم (MSB - پرارزش‌ترین بایت) از آدرس بعدی خوانده می‌شود.

- **ترکیب دو بایت برای بازیابی مقدار int**

```
return (highByte << 8) | lowByte;
```

در این قسمت مقدار highByte را ۸ بیت به چپ شیفت می‌دهد تا در جایگاه ۸ بیت بالا قرار بگیرد. مقدار lowByte را با عملگر | (OR) به مقدار highByte اضافه می‌کند تا مقدار کامل int بازسازی شود.

۷-۲-۱۸. معرفی تابع handleRoot

```
// Manage the Webserver home requests
void handleRoot();
```

این تابع وظیفه مدیریت درخواست‌های صفحه اصلی وب سرور را بر عهده دارد.

- **بررسی وجود فایل در LittleFS**

```
if (!LittleFS.exists("/index.html")) {
  webServer.send(404, "text/plain", "File not found");
  Serial.println("File not found");
  return;
}
```

تابع LittleFS.exists("/index.html") بررسی می‌کند که آیا فایل index.html در سیستم فایل LittleFS موجود است یا نه. اگر فایل وجود نداشته باشد، سرور پاسخ ۴۰۴ (Not Found) ارسال می‌کند و پیغام "File not found" در مانیتور سریال نمایش داده می‌شود. سپس تابع با return از اجرا خارج می‌شود.

- **باز کردن و ارسال فایل**

```
File file = LittleFS.open("/index.html", "r");
webServer.streamFile(file, "text/html");
file.close();
```

ابتدا فایل index.html برای خواندن "r" باز می‌شود. سپس محتوای فایل از طریق webServer.streamFile(file, "text/html") به کلاینت ارسال می‌شود. در نهایت، فایل با file.close() بسته می‌شود تا منابع حافظه آزاد شوند.