



برنامه‌نویسی ربات‌های Lego MINDSTORMS با استفاده از نرم‌افزار LabVIEW

پویان نیّری



فهرست مطالب

فصل اول: آماده‌سازی نرم‌افزار LabVIEW

۲	نصب نرم‌افزار LabVIEW 2016
۶	نصب ماژول LEGO MINDSTORMS
۸	فعال‌سازی نرم‌افزار LabVIEW
۹	مراحل قرار دادن نرم‌افزار در فایروال ویندوز

فصل دوم: مقدمات برنامه‌نویسی با نرم‌افزار LabVIEW

۱۲	مقدمه‌ای بر نرم‌افزار LabVIEW
۱۳	شروع ساخت برنامه با LabVIEW
۱۳	پنجره‌های Front Panel و Block Diagram
۱۴	آشنایی با محیط VI
۱۶	ساخت یک برنامه‌ی ساده
۱۹	نوع داده (Data Type)
۲۰	داده‌های عددی
۲۰	توابع عددی
۲۱	داده‌های دودویی یا باینری
۲۳	داده‌های آرایه‌ای
۲۳	آرایه‌ی یک بعدی
۲۵	آرایه ثابت
۲۶	ساخت آرایه چند بعدی
۲۷	توابع آرایه‌ها
۲۷	ابعاد آرایه
۲۹	فراخوانی درایه
۲۹	جایگزین کردن آرایه
۲۹	سایر توابع
۳۰	ساختارها و حلقه‌ها
۳۰	حلقه For

۳۱	حلقه‌ی For با بلوک شرط
۳۱	حلقه While
۳۲	شیفت رجیستر (Shift Register)
۳۳	ساختار Case Structure
۳۴	ساختار Flat Sequence
۳۵	تمرین فصل دوم

فصل سوم: برنامه‌نویسی MINDSTORMS

۳۷	مقدمه‌ای بر ماژول MINDSTORMS
۴۶	اجزای MINDSTORMS EV3
۴۷	بریک (Brick)
۵۰	موتورها
۵۰	فرمان دادن به موتور
۵۰	فرمان با بلوک Move Motors
۵۴	فرمان با بلوک Fixed Distance
۵۶	بلوک LEGO Steering
۵۷	بلوک Stop Motors
۵۷	کاربرد موتورها
۵۸	(۱) حرکت در مسیر مستقیم
۵۹	(۲) چرخش حول یک چرخ
۶۲	(۳) مسیرهای ترکیبی
۶۴	سنسور تماس
۶۴	خواندن سنسور تماس
۶۶	کاربردهای سنسور تماس
۷۱	سنسور اولتراسونیک
۷۱	خواندن سنسور اولتراسونیک
۷۲	کالیبراسیون سنسور اولتراسونیک
۷۵	کاربردهای سنسور اولتراسونیک

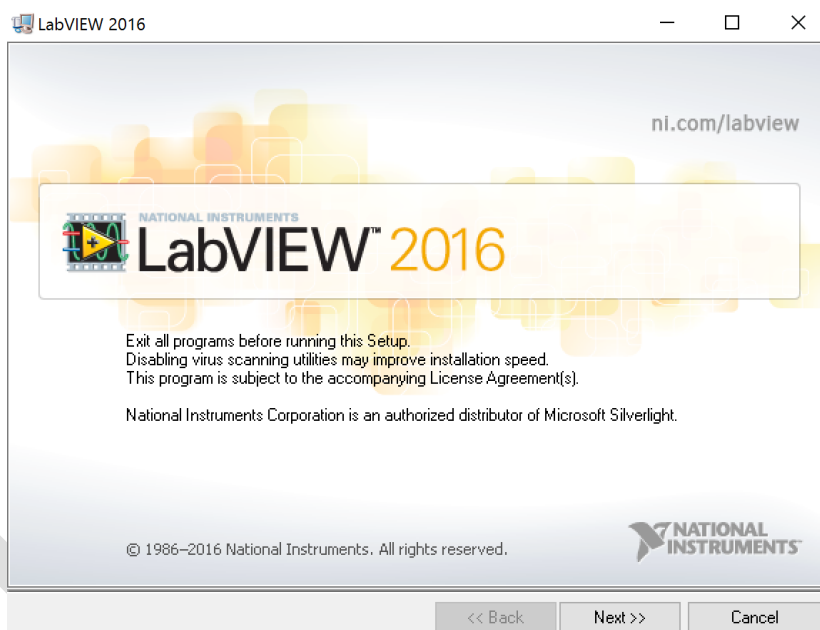
۸۱	سنسور ژيروسکوپ.....
۸۱	خواندن سنسور ژيروسکوپ
۸۲	کالیبراسیون سنسور ژيروسکوپ
۸۳	کاربردهای سنسور ژيروسکوپ.....
۸۹	سنسور رنگ/نور
۸۹	خواندن سنسور رنگ/نور
۹۲	کالیبراسیون سنسور رنگ/نور
۹۳	کاربردهای سنسور رنگ/نور
۹۶	الگوریتم ساده
۹۸	الگوریتم تناسبی
۱۰۰	تمرین فصل سوم
فصل چهارم: برنامه‌نویسی پیشرفته MINDSTORMS	
۱۰۴	مقدمه
۱۰۵	ربات تفکیک رنگ (Color Sorter)
۱۱۵	تمرین ربات تفکیک رنگ
۱۱۶	بازوی رباتیک Robot Arm
۱۱۸	کنترل چرخش حول محور Z
۱۲۳	چرخش حول محور Y
۱۲۶	حرکت گریپر
۱۲۸	برنامه نهایی بازوی رباتیک
۱۲۹	تمرین بازوی رباتیک

فصل اول

«آماده‌سازی نرم‌افزار LabVIEW»

نصب نرم افزار LabVIEW 2016

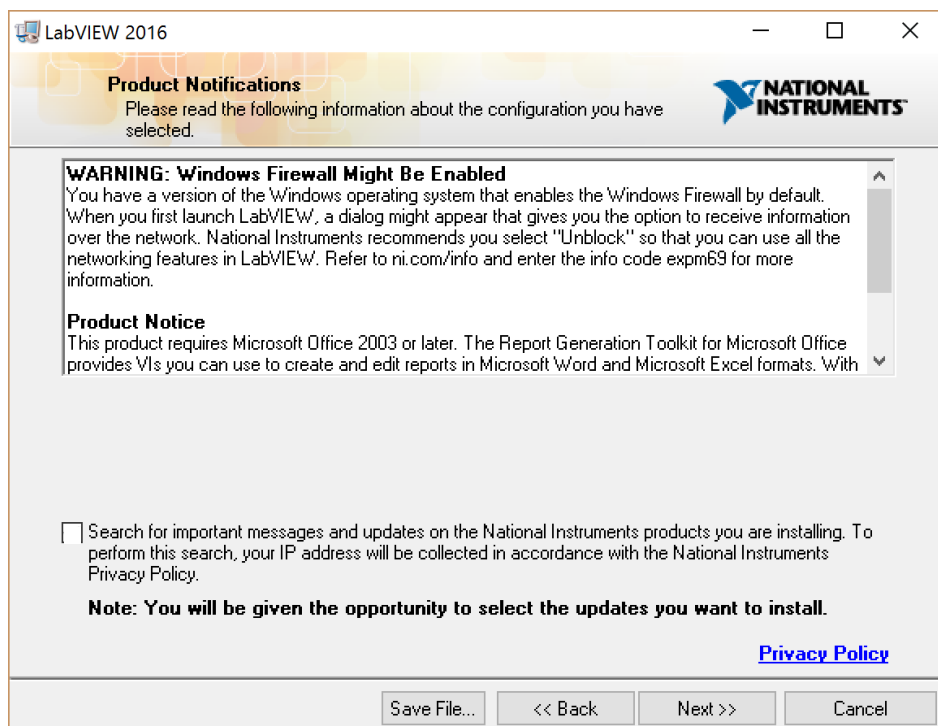
ابتدا به فولدر حاوی نصب نرم افزار LabVIEW 2016 رفته و فایل autorun.exe را اجرا کرده و از آنجا مورد اول برای نصب نرم افزار LabVIEW را انتخاب می کنیم. با انتخاب این گزینه، پنجره ی زیر باز می شود:



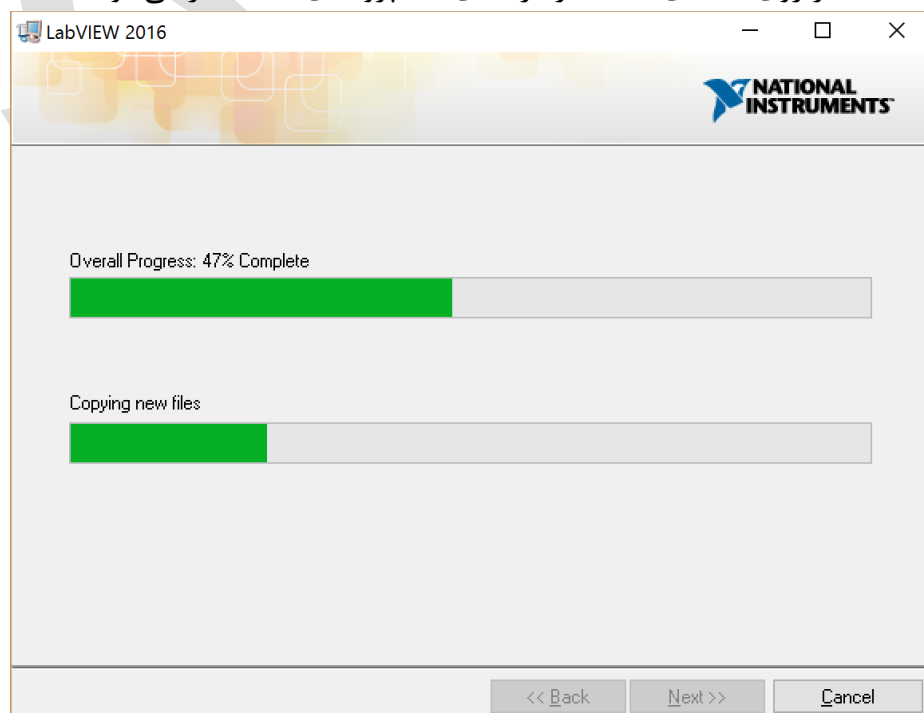
بر روی دکمه ی Next کلیک می کنیم. در مرحله ی بعد اگر مایل هستید نام و نام شرکت خود را ثبت کرده و بر روی دکمه ی Next کلیک کنید.

در پنجره ی بعد شماره سریال های نرم افزار درخواست می شود، کلیه ی فیلدها را خالی گذاشته و بر روی Next کلیک می کنیم. در قسمت بعد محل نصب نرم افزار را مشخص می کنیم. پیشنهاد می شود در مسیر پیش فرض

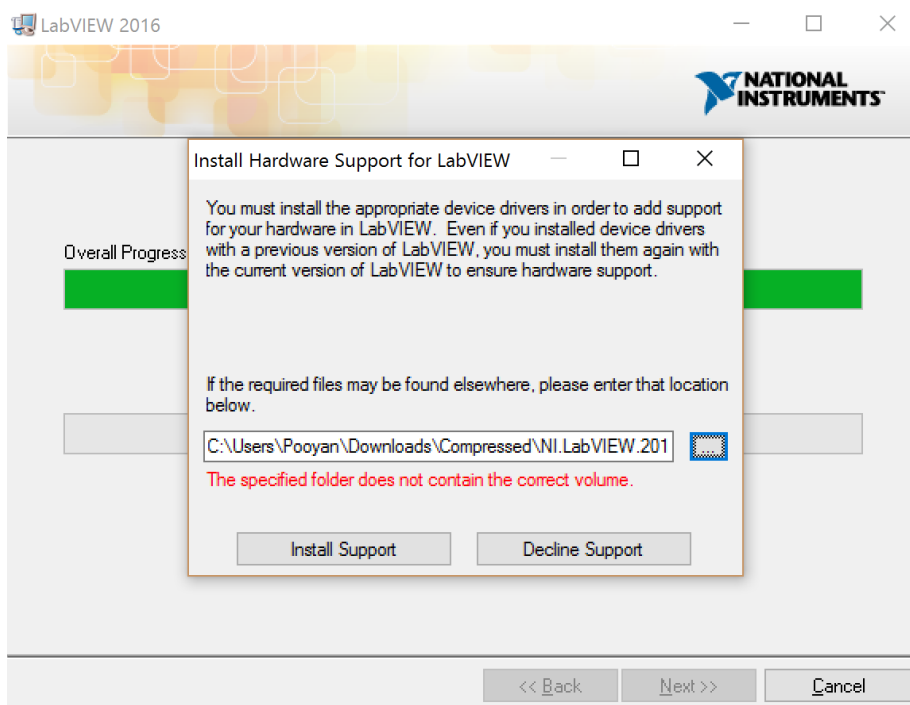
نصب شود. بعد از کلیک بر روی Next، می‌توان اجزای مختلف نرم‌افزار را جهت نصب انتخاب کرد. در حالت پیش فرض تمامی گزینه‌ها فعال است. در قسمت بعد حتما تیک گزینه‌ی Search for important messages and ... را بردارید و بر روی Next کلیک کنید:



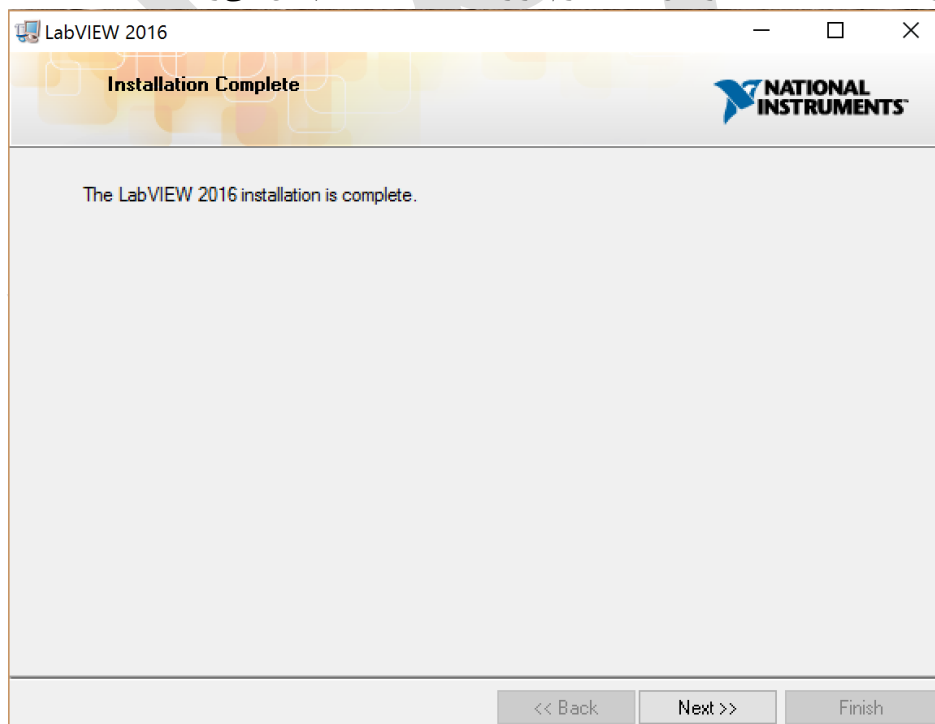
در دو مرحله‌ی بعد گزینه‌ی I accept the above 2 License Agreement(s) را انتخاب کرده و بر روی Next کلیک کنید. با کلیک بر روی دکمه‌ی Next در مرحله‌ی بعد پروسه‌ی نصب آغاز می‌شود:



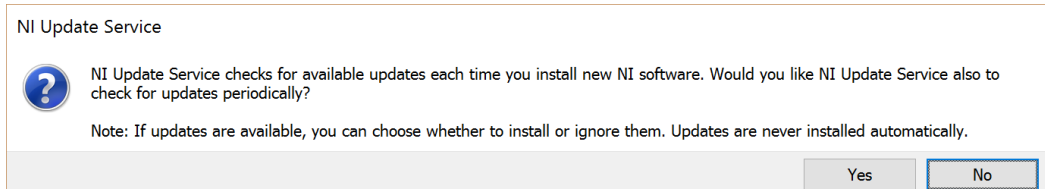
در پایان نصب در صورتی که گزینه‌ی NI Device Drivers را برای نصب انتخاب کرده باشید از شما محل فایل‌های نصب درخواست می‌گردد. در صورتی که قصد نصب آنها را ندارید بر روی دکمه‌ی Decline Support کلیک کنید (برای برنامه‌ریزی ربات‌های MINDSTORMS نیازی به این مورد نمی‌باشد):



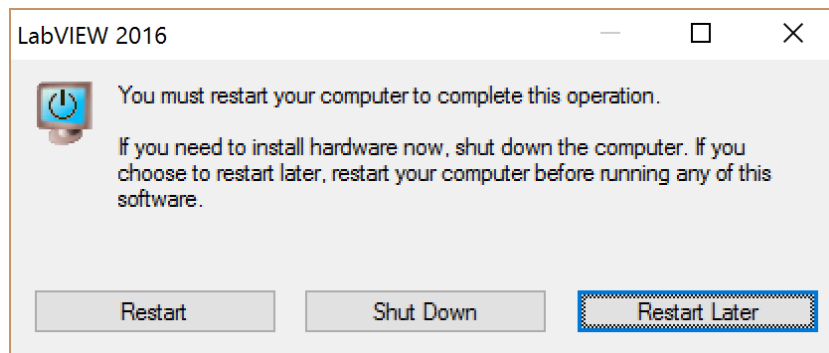
با کلیک بر روی دکمه‌ی Next مراحل نصب نرم‌افزار LabVIEW به پایان می‌رسد:



در صورت مشاهده‌ی پیغام زیر بر روی No کلیک کنید:



در نهایت از شما خواسته می‌شود تا برای تکمیل مراحل نصب، کامپیوتر خود را راه‌اندازی مجدد نمایید. به این دلیل که ما قصد نصب ماژول Lego MINDSTORMS را داریم، در اینجا گزینه‌ی Restart Later را انتخاب می‌کنیم:

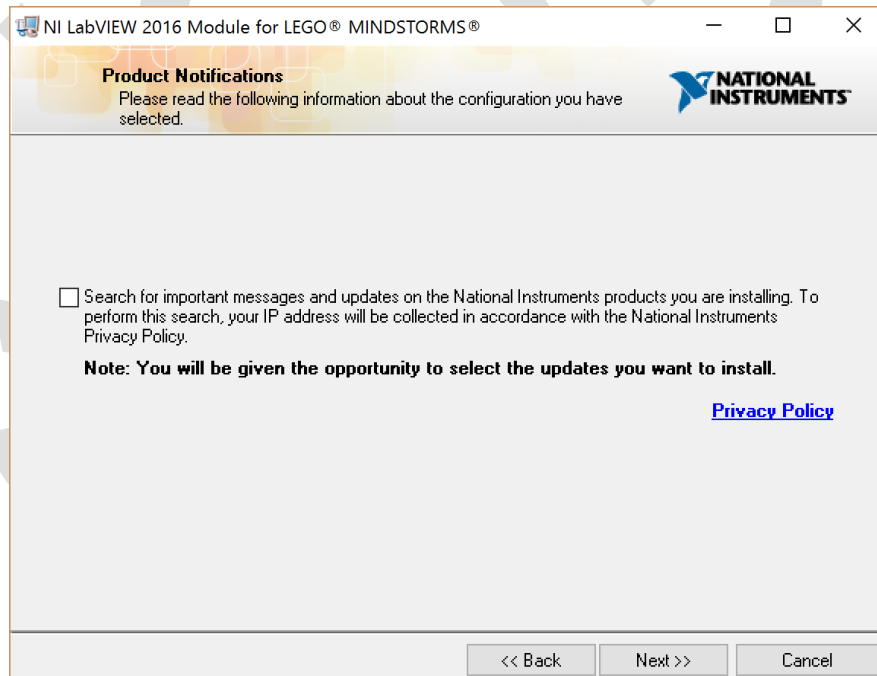


نصب ماژول LEGO MINDSTORMS

با اجرای فایل autorun.exe پنجره‌ی زیر باز می‌شود، بر روی گزینه‌ی ... Install NI LabVIEW ... کلیک کنید:

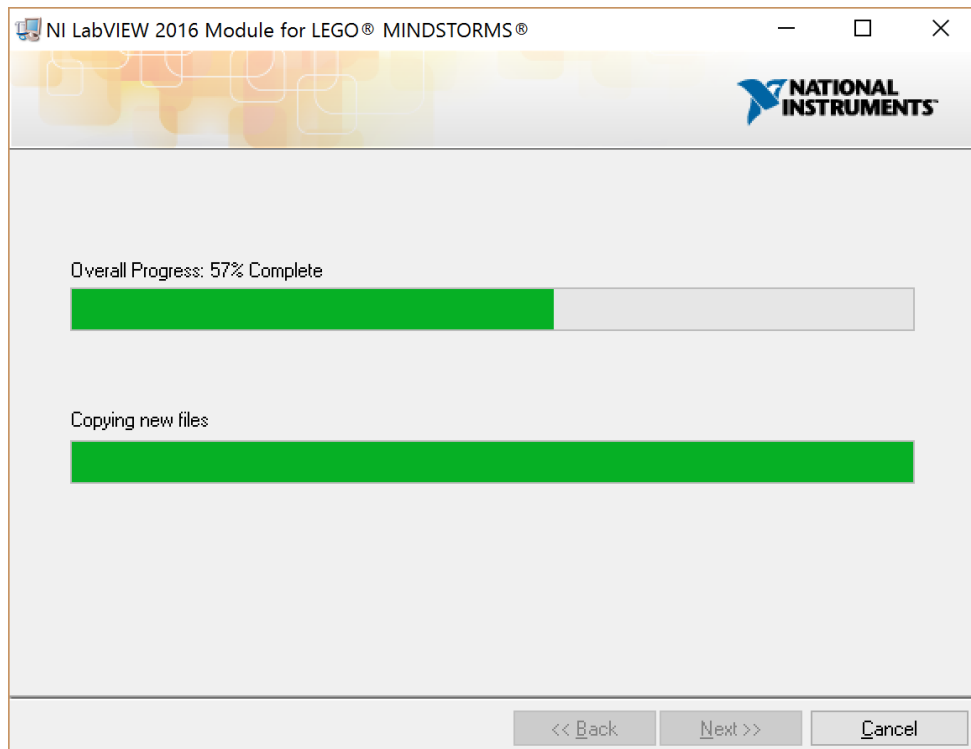


با باز شدن این پنجره مراحل نصب آغاز می‌گردد. بر روی Next کلیک کنید. در قسمت بعد تیک گزینه‌ی Search for important messages and ... را بردارید و بر روی Next کلیک کنید:

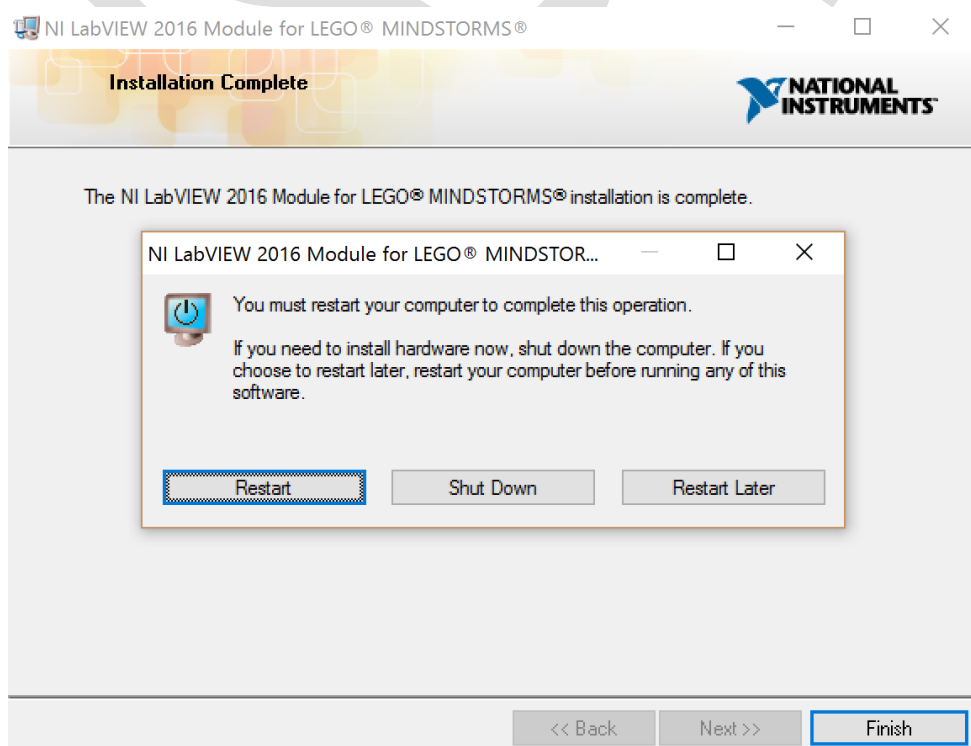


در مرحله‌ی بعد گزینه‌ی I accept the License Agreement را انتخاب کرده و بر روی Next کلیک کنید.

کلیک بر روی دکمه‌ی Next در مرحله‌ی بعد، نصب برنامه آغاز می‌گردد:

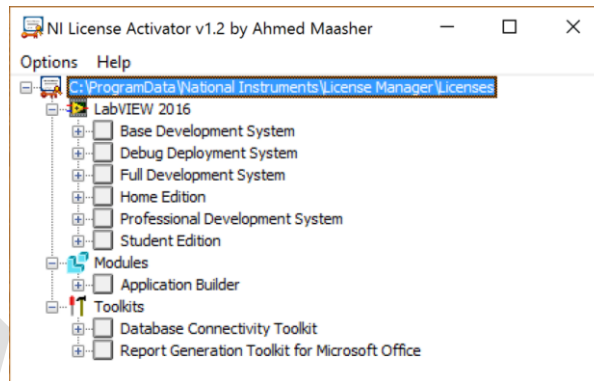


در نهایت با کلیک بر روی گزینه‌ی Finish مراحل نصب ماژول Lego MINDSTORMS به پایان می‌رسد. گزینه‌ی Restart را از پیغام نمایش داده شده انتخاب کنید تا کامپیوتر شما مجدداً راه اندازی شود:

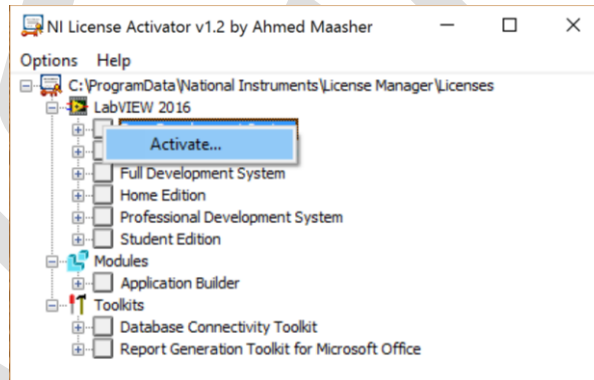


فعال سازی نرم افزار LabVIEW

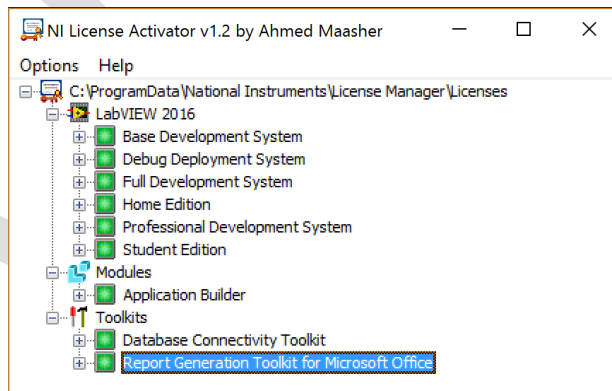
بعد از راه اندازی مجدد کامپیوتر از فولدر Keygen فایل NI License Activator v1.2.exe را اجرا کنید. در صورتی که نرم افزار را در محل پیش فرض نصب کرده باشید، به طور خودکار شناسایی می شود، در غیر این صورت محل نصب نرم افزار را مشخص کنید تا به شکل پنجره ی زیر، اجزای مختلف برنامه شناسایی شود:



با کلیک راست بر روی هر مورد و انتخاب گزینه ی Activate، هر کدام از اجزای نرم افزار فعال سازی می شود:



تمامی اجزای نرم افزار را مطابق شکل زیر فعال سازی نمایید:

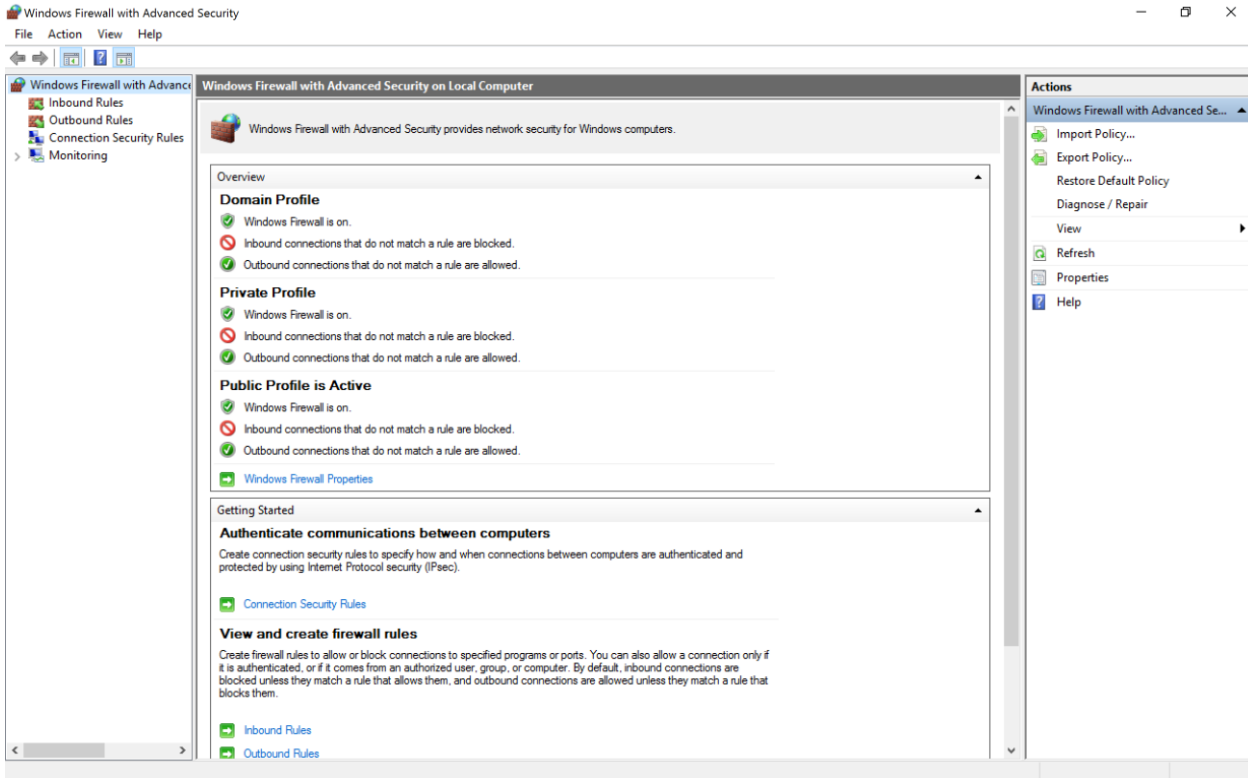


حال مراحل نصب و فعال سازی نرم افزار LabVIEW و ماژول Lego MINDSTORMS به اتمام رسیده است. برای فعال ماندن نرم افزار لازم است تا به وسیله ی فایروال، دسترسی آنرا به اینترنت مسدود نماییم. *

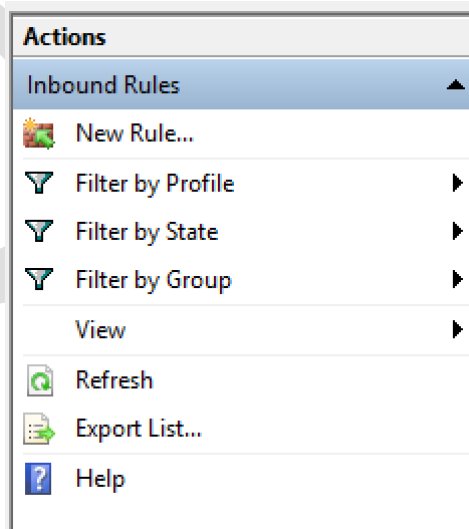
مراحل قرار دادن نرم افزار در فایروال ویندوز

در منوی Start عبارت Firewall را جستجو کرده و گزینه‌ی Windows Firewall and Advanced Security

را باز می‌نماییم:

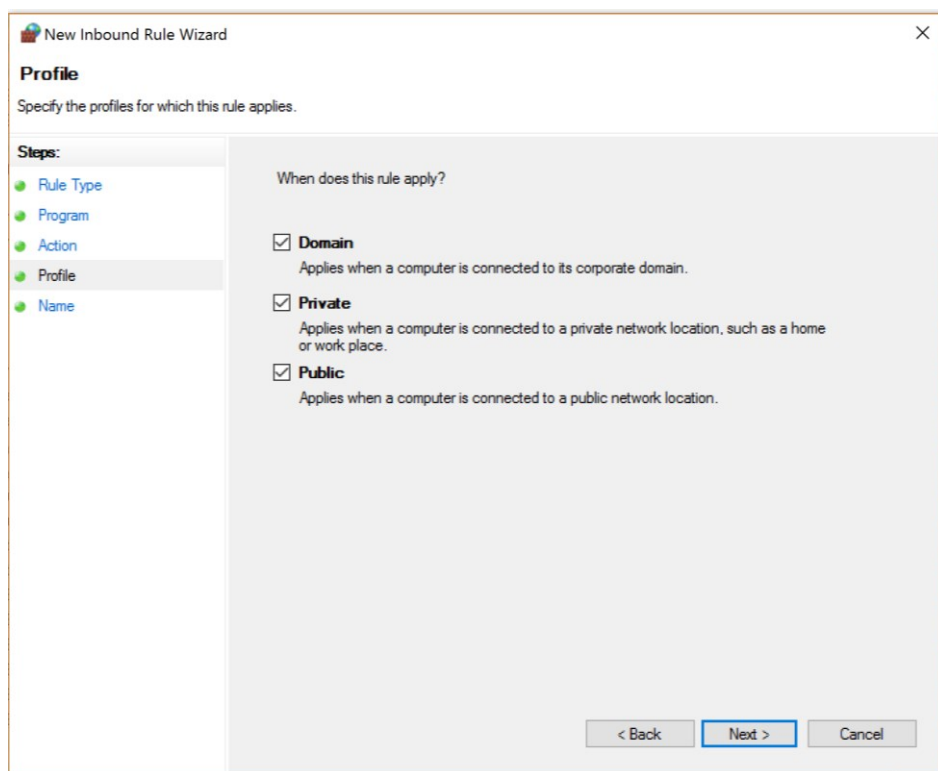


از قسمت سمت چپ بر روی Inbound Rules کلیک کرده و در منوی سمت راست مطابق شکل زیر بر روی New Rule... کلیک کنید:



در مرحله‌ی اول گزینه‌ی Program را انتخاب کنید و Next را بزنید. در قسمت بعد از محل نصب برنامه، فایل LabVIEW.exe را انتخاب می‌کنیم و بر روی Next کلیک می‌کنیم. سپس در مرحله‌ی بعد گزینه‌ی Block the connection را علامت زده و Next را انتخاب می‌کنیم.

در مرحله‌ی بعد مطابق شکل گزینه‌ها را انتخاب کرده و سپس نامی دلخواه انتخاب کرده و Finish را انتخاب می‌نماییم:



همین مراحل را با انتخاب قسمت Outbound Rules از منوی سمت راست، تکرار کرده تا دسترسی نرم‌افزار به اینترنت به طور کامل مسدود گردد. حال نرم‌افزار LabVIEW آماده‌ی اجرا و استفاده است.

فصل دوم

«مقدمات برنامه‌نویسی با نرم‌افزار LabVIEW»

مقدمه‌ای بر نرم‌افزار LabVIEW



نرم‌افزار LabVIEW ساخته شده توسط شرکت National Instruments می‌باشد. این نرم‌افزار با زبان برنامه‌نویسی گرافیکی (به نام زبان G¹ شناخته می‌شود)، در عین سادگی، امکان خلق برنامه‌های پیچیده را به کاربر از جمله مهندسان می‌دهد. نام این برنامه در واقع مخفف عبارت **Laboratory Virtual Instrument Engineering Workbench** است. هدف اصلی توسعه‌ی این نرم‌افزار برای طراحی و ساخت سیستم‌های کنترل و اندازه‌گیری توسط مهندسان و دانشمندان می‌باشد.

به هر برنامه‌ای که در نرم‌افزار LabVIEW نوشته شود VI یا **Virtual Instrument** به معنی ابزار مجازی گفته می‌شود که هر VI شامل سه قسمت **Block Diagram**، **Front Panel** و **Connector Panel** می‌باشد. **Front Panel** یا همان رابط کاربری در برگیرنده‌ی تمامی کنترل‌ها (**Controls**) و شاخص‌های (**Indicators**) برنامه می‌باشد که کنترل‌ها عبارت از ورودی‌هایی است که کاربر به برنامه می‌دهد و شاخص‌ها عبارت از خروجی‌هایی است که به شکل اعداد یا علائم نمایش داده می‌شود.

در پشت صحنه‌ی این پنل، برنامه‌ی اصلی یا همان دیاگرام بلوکی (**Block Diagram**) می‌باشد که بخش اصلی VI می‌باشد. در این بخش، توابع و فرمان‌های مختلف به صورت بلوک‌هایی به یکدیگر متصل می‌شوند تا روند و هدف برنامه را مشخص کنند. تمامی کنترل‌ها و شاخص‌هایی که در **Front Panel** وجود دارند دارای لینک‌هایی در بخش **Block Diagram** می‌باشند که به آنها ترمینال (**Terminal**) گفته می‌شود و از این طریق با پیکره‌ی برنامه در ارتباط هستند. بلوک‌های موجود در این قسمت، ورودی را از **Front Panel** دریافت کرده و بعد از پردازش آنها، نتیجه را به **Front Panel** منتقل می‌کنند.

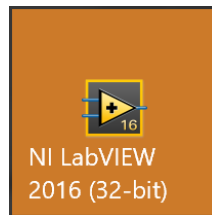
در نهایت در بخش **Connector Panel** یا تابلوی اتصالات می‌توان از یک VI به عنوان یک بلوک در **Block Diagram** سایر VI‌ها استفاده کرد.

در واقع می‌توان گفت یک VI یا ابزار مجازی مانند یک ابزار واقعی است. برای مثال یک کامپیوتر را در نظر بگیرید. تمامی دکمه‌هایی که بر روی صفحه کلید و ماوس وجود دارد و هر ورودی دیگر این کامپیوتر، همان **Front Panel** است که مستقیماً در اختیار کاربر قرار گرفته است. همچنین صفحه نمایش، بلندگو و چراغ‌هایی که در کامپیوتر وجود دارد نیز جزئی از **Front Panel** محسوب می‌شوند چرا که خروجی کامپیوتر را به کاربر نشان می‌دهد. سایر قطعات این کامپیوتر که از دید کاربر مخفی می‌باشد و کاربر به طور مستقیم با آنها سر و کار ندارد و تنها پردازش و عملیات ریاضی و منطقی را انجام می‌دهند مانند مادربرد، کارت گرافیک، رم، CPU و ... متعلق به بخش **Block Diagram** می‌باشد. همچنین اگر این کامپیوتر را به دستگاه یا کامپیوتر دیگری متصل کنیم در واقع ورودی و خروجی این کامپیوتر را در اختیار سایر کامپیوترها قرار داده‌ایم که این عمل وظیفه‌ی **Connector Panel** می‌باشد. پس همانطور که تمامی ابزار فیزیکی به نوعی شامل این سه بخش می‌باشند، هر VI نیز شامل این سه بخش است.

¹ با زبان برنامه‌نویسی g-code تفاوت دارد.

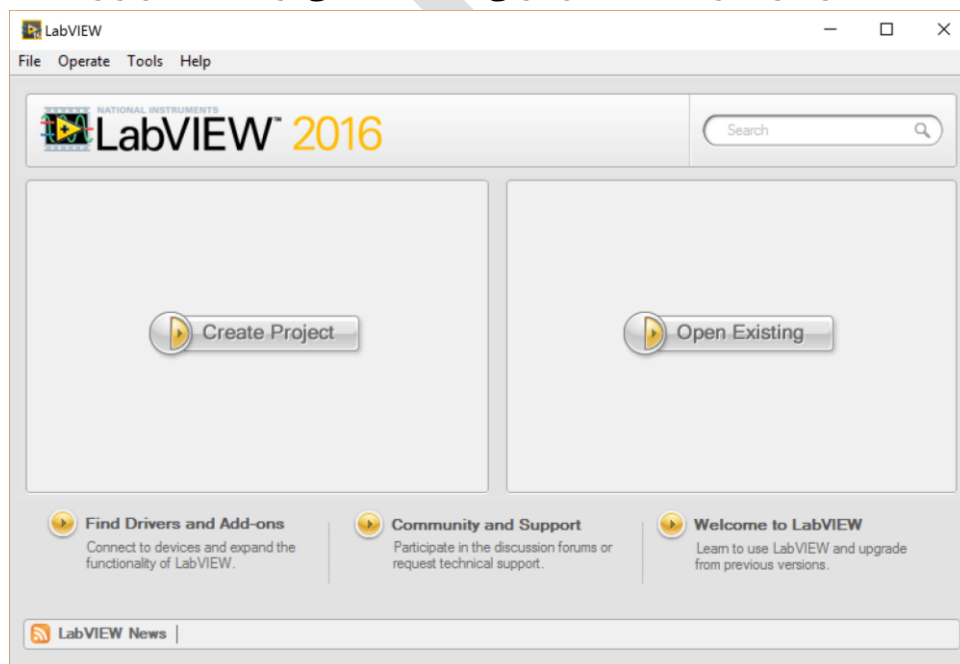
شروع ساخت برنامه با LabVIEW

برای شروع کار با برنامه‌ی LabVIEW، بعد از نصب نرم‌افزار بر روی سیستم خود، از میانبر ایجاد شده بر روی دسکتاپ یا منوی Start اقدام به اجرای برنامه می‌نماییم:



شکل ۱ آیکون نرم‌افزار LabVIEW

اولین پنجره‌ای که بعد از اجرای برنامه به نمایش درمی‌آید صفحه اصلی برنامه به شکل زیر است:



شکل ۲ صفحه‌ی اصلی نرم‌افزار

در این پنجره می‌توان پروژه‌ی جدیدی را تعریف کرد یا پروژه‌های قبلی را مجدداً باز نمود. هر پروژه می‌تواند دربرگیرنده‌ی چند VI باشد. برای شروع ساخت برنامه از منوی بالا به مسیر **File → New** در **Targeted VI** → **NXT/EV3** رفته و بر روی آن کلیک کنید. حال دو پنجره‌ی **Front Panel** و **Block Diagram** مربوط به VI ایجاد شده به نمایش درمی‌آید.

پنجره‌های Front Panel و Block Diagram

در پنجره‌ی مربوط به **Front Panel** همان رابط گرافیکی کاربر یا **Graphical User Interface** به نمایش در می‌آید. همانطور که از اسم آن مشخص است در این قسمت با استفاده از ابزاری که تعبیه شده است می‌توان از کاربر ورودی‌هایی دریافت نمود و خروجی‌هایی را به او نمایش داد. ورودی‌ها و خروجی‌ها می‌توانند به فرم عددی، باینری (دودویی)، رشته‌ای و ... باشند. اصطلاحاً در این نرم‌افزار به ورودی‌هایی که از کاربر گرفته

می‌شوند کنترل (Control) و به خروجی‌هایی که به کاربر نمایش داده می‌شوند شاخص (Indicator) گفته می‌شود.

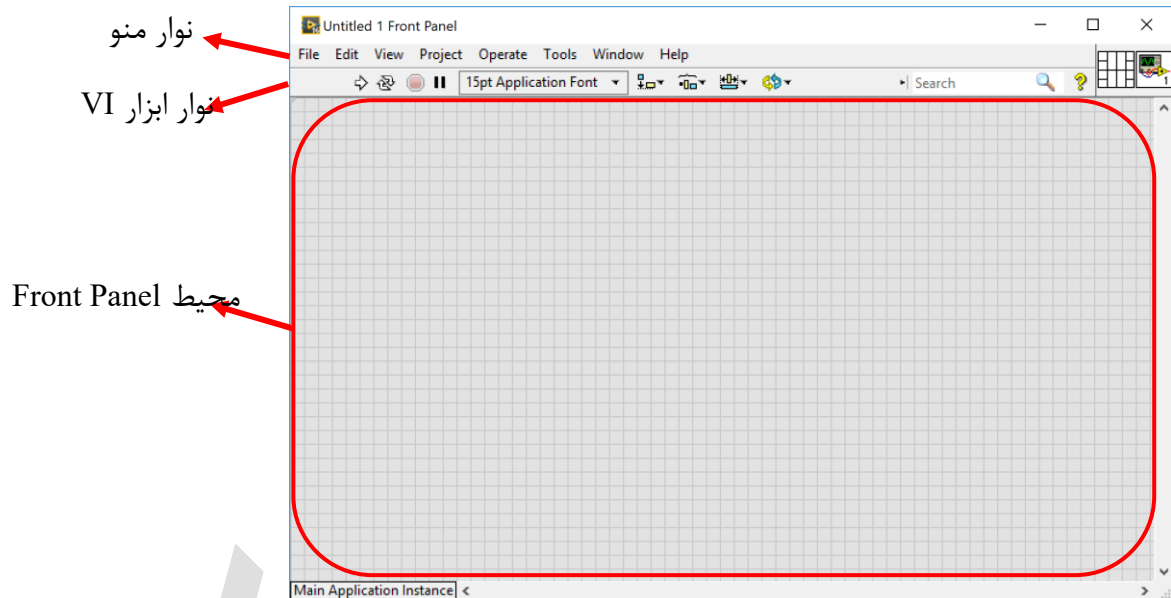
همانطور که گفته شد پنجره‌ی Front Panel تنها رابطی بین کاربر و برنامه است، بنابراین برای ساخت برنامه و طراحی آن باید در پنجره‌ی Block Diagram اقدام به طراحی نمود.

پنجره‌ی Block Diagram یا همان دیاگرام بلوکی، فضایی برای تعیین روند و ساختار برنامه می‌باشد. در این محیط با استفاده از بلوک‌هایی که در اختیار کاربر گذاشته شده و در گروه‌های مختلف مرتب شده‌اند، کاربر اقدام به طراحی ساختار برنامه‌ی خود می‌نماید. این گروه‌ها شامل گروه‌های عددی، آرایه‌ای، منطقی، مقایسه‌ای و ... می‌باشد. با توجه به نوع استفاده از نرم‌افزار، این گروه بندی‌ها تغییر کرده و ابزارهای مورد نیاز کاربر را در اختیار او می‌گذارند. مثلاً در هنگام استفاده از ماژول MINDSTORMS، این نرم‌افزار گروه‌های مرتبط با این هدف را به کاربر نشان می‌دهد. همچنین از طرفی با توجه به ارتباط این بخش با پنجره‌ی Front Panel، برای استفاده از ورودی‌ها و ارسال خروجی‌ها، بلوک‌های ترمینال (Terminal) ایجاد می‌شود که در واقع درگاهی برای انتقال اطلاعات از Front Panel به Block Diagram و بالعکس می‌باشد.

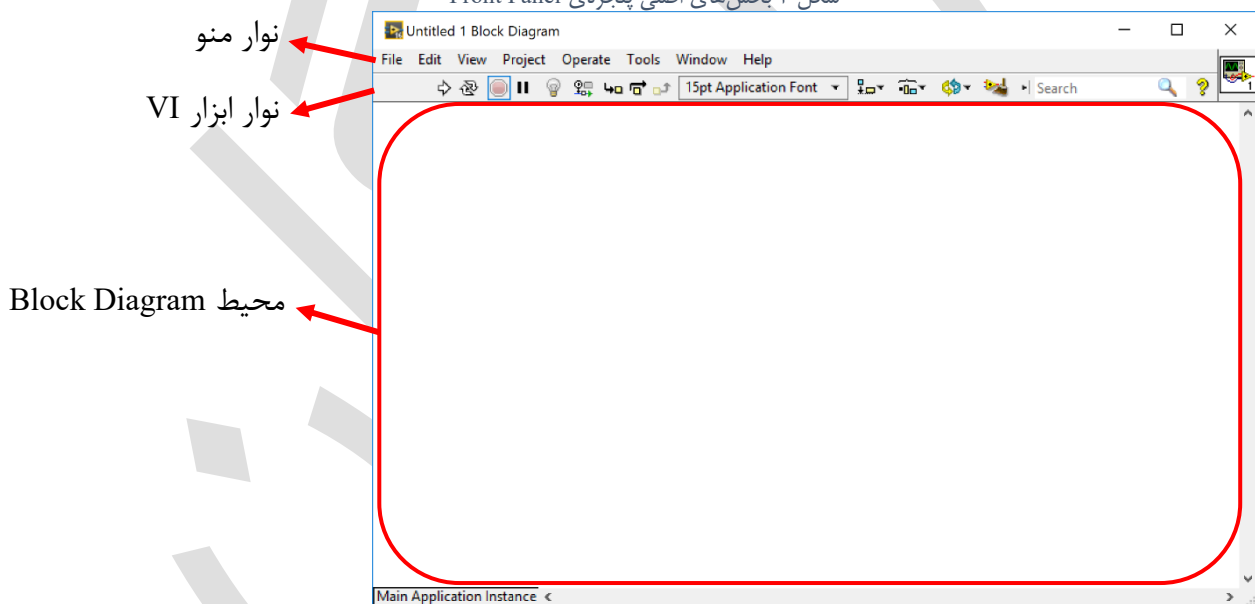
آشنایی با محیط VI

همانطور که گفته شد به هر برنامه‌ی نرم‌افزار LabVIEW، یک ابزار مجازی یا VI گفته می‌شود که خود شامل دو بخش اصلی Front Panel و Block Diagram است. در اینجا می‌خواهیم به طور مختصر با محیط Front Panel و Block Diagram آشنا شویم.

پس از ساخت یک VI جدید در نرم‌افزار، دو پنجره با عنوان‌های Front Panel و Block Diagram به نمایش درمی‌آید. با فشردن کلید ترکیبی Ctrl+/ می‌توانید هر کدام از پنجره‌های Front Panel یا Block Diagram را بر روی صفحه‌ی نمایش بزرگ کنید. همچنین با برای جابجا شدن بین این دو پنجره از کلیدهای ترکیبی Ctrl+E استفاده می‌شود. زمانی که لازم است تا همزمان هر دو پنجره را پیش روی خود داشته باشید کلیدهای ترکیبی Ctrl+T را بفشارید در این صورت هر دو پنجره در کنار یکدیگر قرار خواهند گرفت. چیدمان کلی دو پنجره تا حد زیادی مشابه یکدیگر می‌باشد بنابراین در ادامه به بخش‌های کلیدی و مشترک دو پنجره اشاره خواهیم کرد.



شکل ۴ بخش‌های اصلی پنجره‌ی Front Panel



شکل ۳ بخش‌های اصلی پنجره‌ی Block Diagram


همانطور که مشاهده می‌نمایید این پنجره از چند قسمت تشکیل شده است. نوار بالایی این پنجره مانند اکثر برنامه‌ها، نوار منو نامیده می‌شود که عموماً از آن برای ایجاد، ذخیره سازی یا بارگذاری یک VI استفاده می‌شود. نواری که در پایین نوار منو می‌باشد با نام نوار ابزار VI شناخته می‌شود. از این نوار می‌توان برای اجرا و توقف VI و همچنین مرتب‌سازی بلوک‌ها استفاده نمود.


بخش اعظم این پنجره را محیط Front Panel یا Block Diagram تشکیل می‌دهد. در واقع این محیط همان فضای در دسترس کاربر است که انواع ورودی و خروجی‌ها و یا بلوک‌ها را نشان می‌دهد.

به علت اهمیت زیادی که نوار ابزار دارد، به معرفی کلیدهای آن می‌پردازیم:


اجرا (Run): با کلیک بر روی این دکمه، VI مربوطه اجرا شده و پردازش انجام می‌گردد. این 

گزینه از طریق کلیدهای ترکیبی Ctrl+R نیز قابل دسترسی است.


 **در حال اجرا:** هنگامی که VI در حال اجرا شدن می‌باشد دکمه‌ی Run به این شکل درمی‌آید.


 **خطا در اجرا:** هنگامی که کلید Run به این شکل درمی‌آید به این معنی است که خطایی در VI

وجود دارد، با کلیک بر روی این کلید، لیست خطاها به نمایش درمی‌آید.

 **اجرای مداوم (Run Continuously):** با انتخاب این کلید، تا زمانی که برنامه را متوقف یا لغو

نکنید، به طور مداوم اجرا خواهد شد.

 **لغو عملیات (Abort Execution):** این گزینه اجرای برنامه را لغو می‌کند.

 **توقف (Pause):** روند اجرای برنامه را به طور موقت متوقف می‌نماید. در صورت متوقف بودن

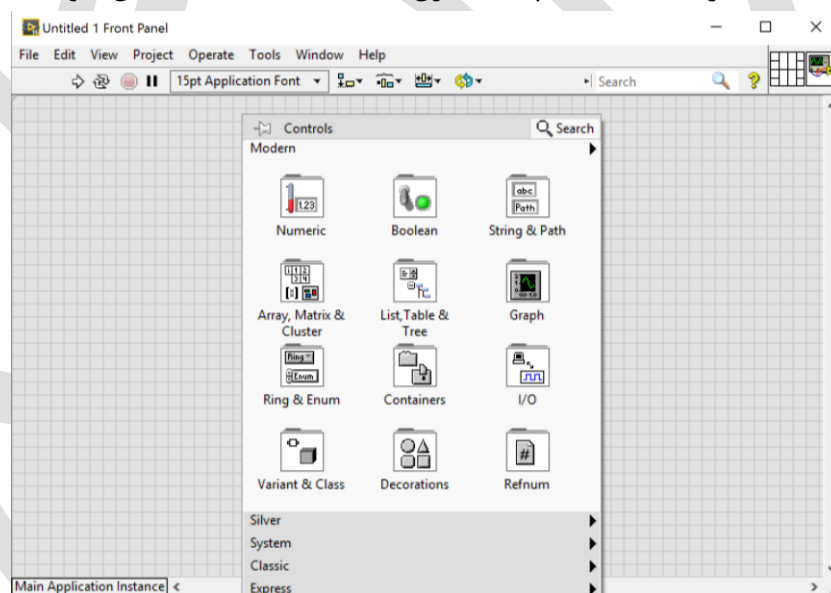
برنامه، با کلیک بر روی این دکمه، برنامه ادامه پیدا خواهد کرد.

ساخت یک برنامه‌ی ساده

برای شروع کار با VI، مثال ساده‌ای را با هم بررسی می‌نماییم.

پس از ساخت اولین VI و به نمایش در آمدن دو پنجره‌ی Front Panel و Block Diagram، در هر نقطه از

محیط Front Panel کلیک راست نمایید تا پالت کنترل (Control Palette) نمایان شود.



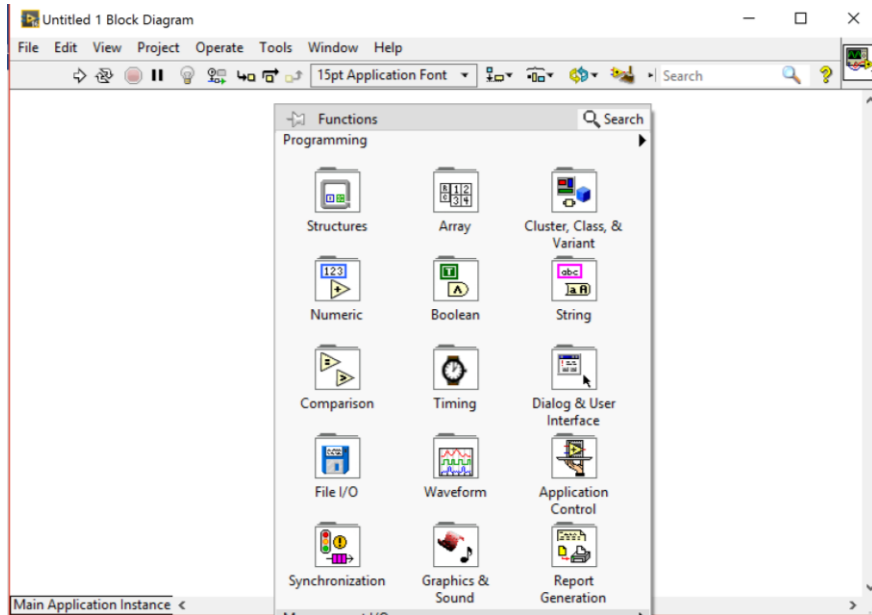
شکل ۵ نمایش پالت کنترل با کلیک راست در پنجره‌ی Front Panel

همانطور که مشاهده می‌کنید گروه‌های مختلف کنترل و شاخص به نمایش درآمده است. در صورتی که نیاز

به جستجو برای یک کنترل یا شاخص داشته باشید می‌توانید از فیلد Search موجود بر روی پالت کنترل

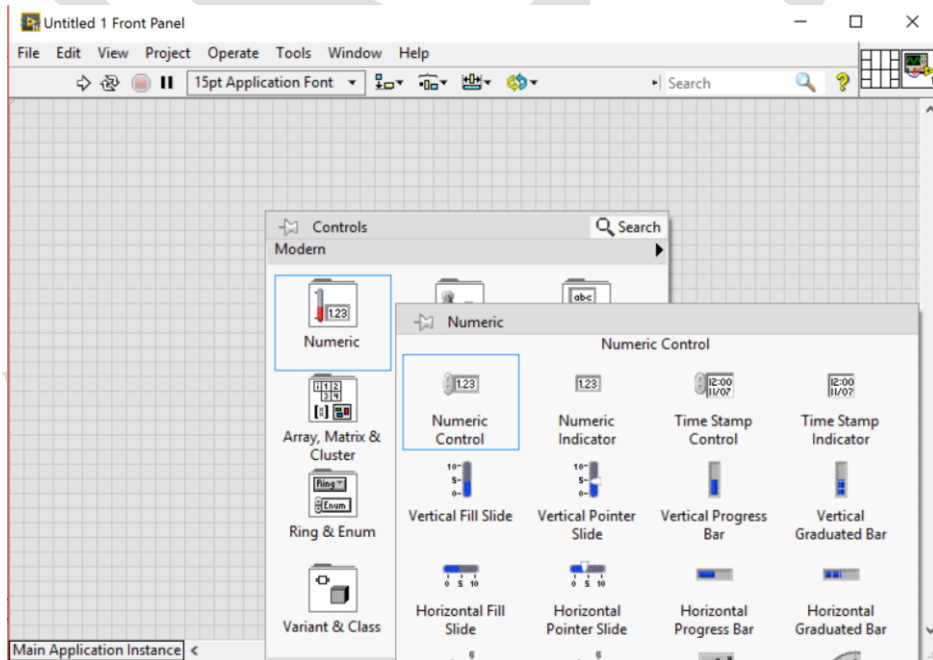
استفاده کنید. اگر همین عمل (کلیک راست) را در محیط Block Diagram انجام دهید خواهید دید که پالت

توابع (Functions Palette) برای شما باز می‌شود که شامل گروه‌های مختلفی می‌باشد:



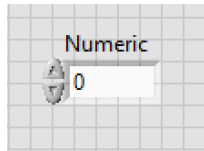
شکل ۶ نمایش پالت توابع با کلیک راست در پنجره‌ی Block Diagram

حال می‌خواهیم در پنجره‌ی Front Panel عددی را به عنوان ورودی از کاربر گرفته و در همان پنجره، همان عدد را به عنوان خروجی نشان دهیم. برای این منظور با استفاده از پالت کنترل، وارد گروه Numeric شده و مورد Numeric Control را انتخاب نمایم:



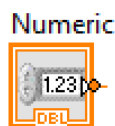
شکل ۷ انتخاب بلوک Numeric Control

با انتخاب این گزینه، می‌توانید کنترل Numeric Control یا همان ورودی عددی را در هر کجای صفحه‌ی Front Panel قرار دهید. با کلیک چپ ماوس این کنترل قرار داده می‌شود:



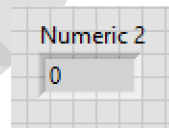
شکل ۸ قرار دادن بلوک Numeric Control در پنجره‌ی Front Panel

عدد نشان داده شده در فیلد این کنترل، همان ورودی است که کاربر به برنامه می‌دهد. با کلیک بر روی فلش‌های کوچک موجود در سمت چپ این فیلد، مقدار این عدد تغییر می‌کند. همچنین می‌توانید با انتخاب کردن عدد و تایپ مقدار مورد نظر به این کنترل مقدار دهی کنید. حال اگر دقت نمایید، در پنجره‌ی Block Diagram نیز بلوکی با عنوان Numeric ایجاد شده است که همان Terminal مرتبط با کنترل ایجاد شده است. با بررسی بیشتر بلوک Numeric می‌توان متوجه شد که این بلوک دارای یک گره می‌باشد:



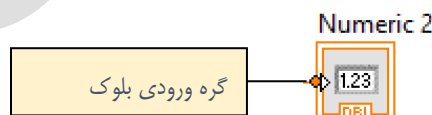
شکل ۹ بلوک Numeric مرتبط با بلوک Numeric Control

این گره در واقع همان عدد وارد شده توسط کاربر است که از طریق این گره می‌توان آنرا وارد فضای Block Diagram نمود. در قدم بعد در پنجره‌ی Front Panel با کلیک راست ماوس و انتخاب گروه Numeric، گزینه‌ی Numeric Indicator را بر می‌گزینیم. با قرار دادن این گزینه در فضای Front Panel، شاخصی با نام Numeric 2 ایجاد می‌شود. این شاخص وظیفه‌ی نمایش اعداد به کاربر را دارد. این عدد می‌تواند هر چیزی باشد و بستگی به برنامه و دیاگرام بلوکی آن برنامه دارد.



شکل ۱۰ بلوک Numeric Indicator در پنجره‌ی Front Panel

مجدداً مشابه قبل، برای این شاخص نیز یک بلوک در پنجره‌ی Block Diagram ایجاد شده است. این بلوک نیز دارای یک گره می‌باشد که در واقع خروجی‌ها را از فضای Block Diagram به کاربر در فضای Front Panel نمایش می‌دهد:



شکل ۱۱ بلوک Numeric مرتبط با بلوک Numeric Indicator

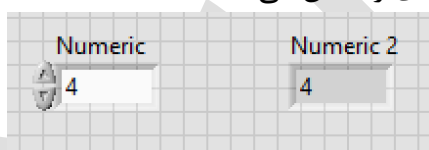
اما اگر کمی با دقت بیشتر بررسی کنیم دو بلوک Numeric و Numeric 2 با اینکه هر کدام یک گره دارند اما تفاوتی با یکدیگر ندارند. در بلوک Numeric که مربوط به کنترل یا ورودی عددی می‌باشد گره‌ی موجود به صورت خروجی می‌باشد که می‌توان از این گره برای ورودی سایر بلوک‌ها استفاده کرد. اما در بلوک Numeric 2 که همان ترمینال شاخص یا خروجی عددی می‌باشد گره‌ی موجود به صورت ورودی می‌باشد که با اتصال هر عددی به این گره، آن عدد به کاربر نمایش داده می‌شود.

برای نمایش عدد ورودی در خروجی، کفایست تا در فضای Block Diagram، گرهی خروجی بلوک Numeric را به گرهی ورودی بلوک Numeric 2 متصل نماییم. برای این منظور بر روی گرهی خروجی Numeric کلیک کرده تا سیمی از این بلوک ایجاد شود، سپس بر روی گرهی ورودی Numeric 2 کلیک نمایید تا این دو بلوک به یکدیگر متصل شوند. با اینکار برنامه ای را طراحی کرده ایم تا عددی را از کاربر دریافت کرده و همان عدد را به کاربر نمایش دهد:



شکل ۱۲ اتصال بلوک شاخص و کنترل عددی به یکدیگر

برای اجرای برنامه لازم است تا آنرا Run کنیم. با اجرا کردن برنامه، از آنجایی که برنامه‌ی طراحی شده بسیار ساده می‌باشد عملیات به سرعت انجام شده و خروجی نشان داده می‌شود. با تغییر مقدار ورودی و اجرای برنامه، خروجی نیز همان مقدار ورودی را نشان می‌دهد:



شکل ۱۳ نمایش عدد ورودی در پنجره‌ی Front Panel

برای اجرای مداوم برنامه و بروزرسانی آنی خروجی، لازم است تا گزینه‌ی **Run Continuously** را کلیک کنید تا برنامه‌ی شما همواره اجرا شود. در این حالت با تغییر ورودی، خروجی نیز تغییر می‌کند و نیازی به اجرای مجدد برنامه نمی‌باشد.

نوع داده (Data Type)

نرم افزار LabVIEW داده‌ها را به انواع مختلفی در حافظه ذخیره می‌نماید. از مهمترین انواع داده‌ها می‌شود به موارد زیر اشاره کرد:

- **داده‌های دودویی (Boolean):** این نوع داده‌ها تنها می‌توانند دو حالت داشته باشند: مقدار صفر یا FALSE و هر مقدار غیر صفر یا TRUE. این نوع داده‌ها به صورت مقادیر ۸ بیتی در حافظه ذخیره می‌شوند.
- **داده‌های صحیح (Integer):** این نوع داده‌ها به صورت اعداد صحیح (غیر اعشاری) ذخیره شده که می‌توانند با و بدون علامت (signed و unsigned) بیان شوند. داده‌های Integer بسته به بیت‌های اختصاص داده شده به این داده‌ها دسته‌بندی می‌شوند:

- **Byte Integer:** ۸ بیتی
- **Word Integer:** ۱۶ بیتی
- **Long Integer:** ۳۲ بیتی
- **Quad Integer:** ۶۴ بیتی

- **داده‌های ممیز شناور (Floating-point):** این داده‌ها برای نمایش اعداد حقیقی به روش نماد علمی در کامپیوتر استفاده می‌شوند. داده‌های ممیز شناور در نرم‌افزار LabVIEW با دو دقت قابل ذخیره‌سازی می‌باشند: دقت ساده (Single Precision) و دقت مضاعف (Double Precision).

داده‌های عددی

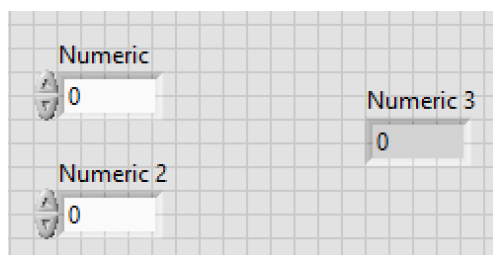
در قدم بعد باید داده‌های وارد شده به برنامه را پردازش کرده و خروجی مطلوب را به کاربر نمایش دهیم. این عملیات می‌تواند عملیات عددی مانند جمع و ضرب و ... یا عملیات منطقی و مقایسه‌ای یا هرگونه عملیات دیگری باشد که خروجی آن بسته به نوع عملیات متفاوت است. همچنین ورودی‌های برنامه می‌تواند از کاربر، سنسور و یا هر طریق دیگری باشد.

توابع عددی

برای پردازش داده‌های عددی، توابعی به صورت پیش‌فرض در نرم‌افزار LabVIEW تعبیه شده است. در ادامه برای آشنایی بیشتر به معرفی این توابع می‌پردازیم:

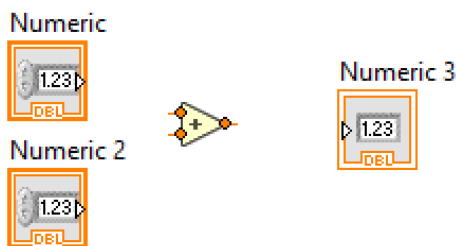
- **جمع:** دو مقدار عددی را با یکدیگر جمع کرده و در گرهی خروجی می‌ریزد.
- **تفریق:** دو مقدار عددی را از یکدیگر کم کرده و در گرهی خروجی می‌ریزد.
- **ضرب:** حاصل ضرب دو مقدار عددی را در گرهی خروجی می‌ریزد.
- **تقسیم:** حاصل تقسیم دو مقدار عددی را در گرهی خروجی می‌ریزد.
- **خارج قسمت و باقیمانده:** خارج قسمت و باقیمانده‌ی تقسیم دو مقدار عددی را در دو گرهی خروجی می‌ریزد.
- **افزایش واحد:** عدد ورودی را به اندازه‌ی یک واحد افزایش می‌دهد.
- **کاهش واحد:** عدد ورودی را به اندازه‌ی یک واحد کاهش می‌دهد.
- **قدرمطلق:** مقدار قدرمطلق عدد ورودی را تولید می‌کند.
- **رادیکال:** ریشه‌ی دوم عدد ورودی را در خروجی می‌ریزد.
- **به توان دو:** عدد ورودی را به توان ۲ می‌رساند.

برای مثال می‌خواهیم برنامه‌ای بنویسیم که دو عدد را از کاربر گرفته و در خروجی، حاصل جمع آن دو عدد را نمایش دهد. مانند قسمت قبل ابتدا دو کنترل عددی و یک شاخص عددی در پنجره‌ی Front Panel قرار می‌دهیم:



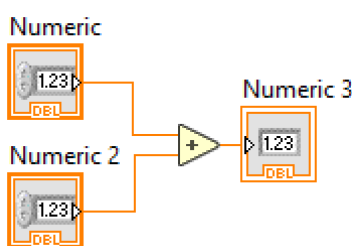
شکل ۱۴ ایجاد دو بلوک کنترل عددی و یک بلوک شاخص عددی

با توجه به اینکه می‌خواهیم دو عدد را با یکدیگر جمع نماییم، در فضای Block Diagram از بلوک Add موجود در پالت Functions و گروه Numeric استفاده می‌نماییم:



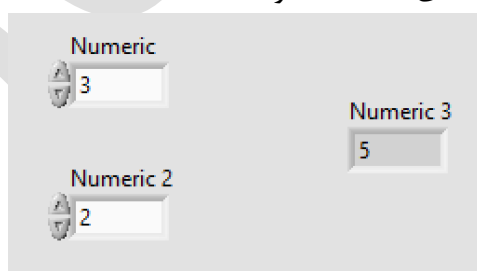
شکل ۱۵ اضافه کردن بلوک جمع در Front Panel

همانطور که مشاهده می‌شود این بلوک دارای سه گره می‌باشد، گره‌های سمت چپ آن ورودی بوده و گره‌ی سمت راست آن خروجی می‌باشد، یعنی با متصل کردن دو مقدار عددی به گره‌های سمت چپ، گره‌ی سمت راست مقدار حاصل جمع آنها را نمایش می‌دهد. پس بلوک‌ها را به شکل زیر متصل می‌نماییم:



شکل ۱۶ اتصال بلوک‌های کنترل و شاخص عددی به بلوک جمع

با اجرای مداوم برنامه (Run Continuously)، می‌توان با وارد کردن هر دو عدد در قسمت کنترل عددی، مقدار حاصل جمع را در شاخص عددی مشاهده نمود:



شکل ۱۷ نمایش جمع دو عدد ورودی در بلوک شاخص عددی

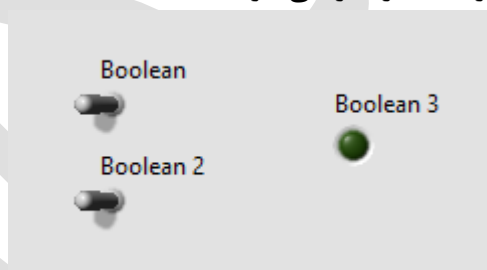
داده‌های دودویی یا باینری

مثالی که در بالا آورده شد مربوط به جمع دو مقدار عددی است. عملیات بر روی داده‌ها می‌تواند به صورت منطقی، مقایسه‌ای و ... انجام شود. برای بررسی نوع دیگری از داده‌ها، داده‌های دودویی یا باینری را در نظر می‌گیریم. این داده‌ها می‌توانند در هر لحظه تنها دو مقدار داشته باشند: ۰ یا ۱. این دو مقدار را می‌توان به شکل‌های مختلفی تلقی نمود، مثلاً می‌توان هر داده باینری را به عنوان چراغی در نظر گرفت که با داشتن مقدار ۰، چراغ خاموش و با داشتن مقدار ۱، چراغ روشن می‌باشد. همچنین اگر بخواهیم به صورت ورودی

در نظر بگیریم، می توان یک کلید خاموش/روشن را در نظر گرفت که با مقدار ۰ در حالت خاموش و با مقدار ۱ در حالت روشن است.

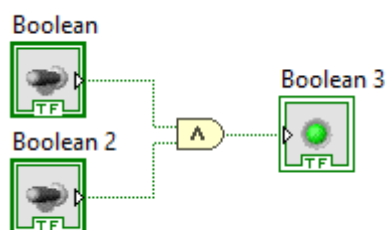
مانند داده‌های عددی که برای آنها عملیات ریاضی مانند جمع و ضرب و ... تعریف می‌شوند، در اینجا هم برای داده‌های دودویی عملیات مشخصی تعریف می‌شود. از مهمترین و اصلی‌ترین عملیات می‌توان به موارد زیر اشاره کرد:

- «و» (AND): این عمل مانند عمل ضرب برای داده‌های دودویی می‌باشد و برای عمل بین دو داده باینری تعریف می‌گردد. جواب این عمل با ضرب دو داده‌ی باینری در یکدیگر بدست می‌آید. مثلاً اگر عملیات ۱ و ۰ را در نظر بگیریم $0 \times 1 = 0$ می‌باشد پس $0 \text{ AND } 1 = 0$. این عملیات در صورتی که هر دو داده ۱ باشند دارای جواب ۱ و در غیر اینصورت مقدار ۰ را به عنوان جواب بدست می‌دهد.
 - «یا» (OR): عمل یا معادل با عمل جمع در داده‌های عددی است. جواب این عمل در صورتی که حداقل یکی از داده‌ها مقدار ۱ داشته باشد برابر با ۱ در غیر اینصورت برابر با ۰ می‌باشد.
 - «نقیض» (NOT): با کمک این عمل می‌توان داده‌ی مورد نظر را عکس کرد. یعنی اگر داده‌ی ما ۱ باشد آنرا ۰ کرده و اگر ۰ باشد آنرا ۱ می‌کند.
- برای مثال می‌خواهیم دو کلید داشته باشیم که در صورت روشن بودن هر دوی آنها، چراغی روشن شود، بنابراین می‌توان مانند شکل زیر برنامه‌ای را طراحی کرد:



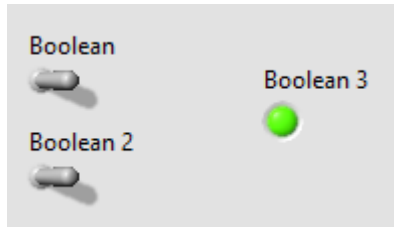
شکل ۱۸ ایجاد دو بلوک کنترل دودویی و یک بلوک شاخص دودویی

که از دو کلید رادیویی به عنوان ورودی برنامه و یک چراغ به عنوان خروجی برنامه استفاده شده است. برای روشن شدن چراغ در مواقعی که هر دو کلید وصل می‌باشد از بلوک AND استفاده می‌کنیم. برای این منظور در پنجره‌ی Block Diagram کلیک راست کرده و از پالت Functions و گروه Boolean، بلوک AND را انتخاب می‌نماییم:



شکل ۱۹ اتصال بلوک‌ها به بلوک AND

با اجرای مداوم برنامه، با تغییر دادن وضعیت کلیدها، چراغ در مواقعی که هر دو کلید وصل می‌باشند روشن می‌شود:



شکل ۲۰ اجرای برنامه‌ی AND منطقی

داده‌های آرایه‌ای

آرایه‌ها بیانگر بردارها یا ماتریس‌ها می‌باشند. مانند بردار یا ماتریس، آرایه نیز شامل چند داده از نوع مشابه می‌باشد. هر آرایه دارای دو قسمت می‌باشد: درایه (یا المان) و ابعاد (سایز آرایه). آرایه‌ها می‌توانند شامل داده‌های عددی، دودویی، رشته‌ای و ... باشند. از آرایه می‌توان هم به عنوان ورودی (کنترل) و هم به عنوان خروجی (شاخص) استفاده کرد.

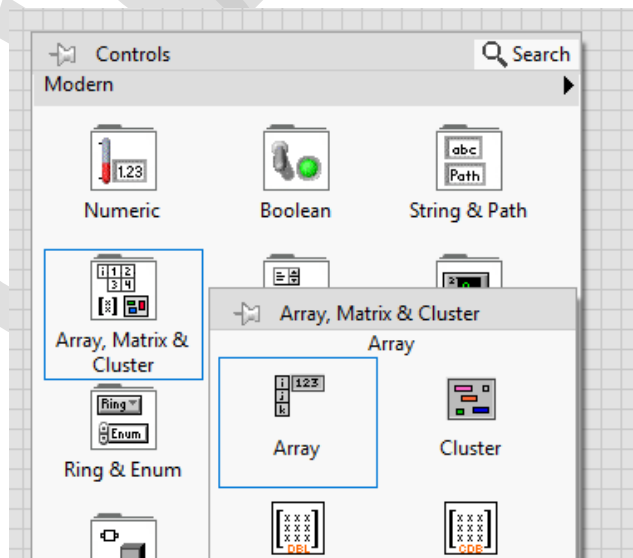
مزیت اصلی آرایه‌ها در مواردی است که تعداد زیادی از داده‌ها با نوع مشابه در دسترس می‌باشد و یا قرار است بر روی چند داده با نوع مشابه، عملیاتی یکسان انجام دهیم.

هر درایه در یک آرایه دارای اندیس مشخصی می‌باشد. اندیس مانند آدرس یک درایه در آرایه است.

* اندیس‌گذاری در آرایه‌ها از صفر شروع می‌شود به این معنی که اندیس درایه‌ی اول مقدار صفر بوده و به همین ترتیب سایر درایه‌ها، اندیس‌گذاری می‌شوند.

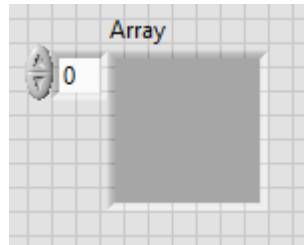
آرایه‌ی یک بعدی

برای ساخت آرایه‌ی یک بعدی در نرم‌افزار LabVIEW، در پنجره‌ی Front Panel، کلیک راست کرده و از پالت Controls، به بخش Array, Matrix & Cluster رفته و Array را انتخاب کنید:



شکل ۲۱ انتخاب بلوک آرایه در پالت کنترل

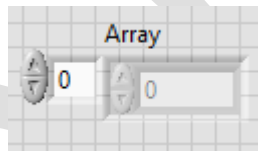
با قرار دادن Array در هر جای صفحه، می‌توان آرایه‌ای را ساخت:



شکل ۲۲ ایجاد بلوک آرایه در Front Panel

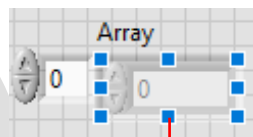
همانطور که مشاهده می‌شود، در سمت چپ آرایه‌ی ایجاد شده یک کنترل عددی قرار دارد که بیانگر اندیس درایه‌ی موجود در خانه‌ی سمت چپ است. در واقع با انتخاب هر مقدار در این کنترل عددی، این آرایه، درایه‌های از این اندیس به بعد را به کاربر نمایش می‌دهد.

همانطور که ذکر شد، آرایه می‌تواند شامل داده‌هایی از نوع عددی، باینری و ... باشد. در ابتدا آرایه‌ی قرار داده شده خالی می‌باشد. در این قسمت با توجه به نوع داده‌ای که مدنظر است می‌تواند یک ثابت عددی، یک متغیر باینری یا ... استفاده کرد تا آرایه‌ی قرار داده شده، از همان نوع تشکیل شود. برای مثال در اینجا می‌خواهیم آرایه‌ای با داده‌های عددی تشکیل دهیم، برای این منظور از پالت کنترل، در قسمت Numeric، یک Numeric Control را به درون آرایه‌ی تشکیل شده اضافه می‌نماییم:



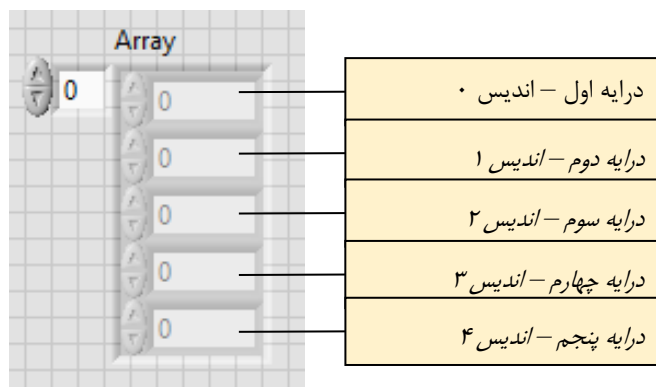
شکل ۲۳ قرار دادن ورودی عددی در آرایه

در این حالت آرایه‌ی تشکیل شده از نوع ورودی (کنترل) بوده و شامل متغیرهای عددی می‌باشد. برای تغییر سایز یا بعد آرایه، می‌توان با نگاه داشتن نشانگر ماوس بر روی کادر محاط بر کنترل عددی و سپس حرکت دادن نقاط آبی نمایان شده استفاده کرد:



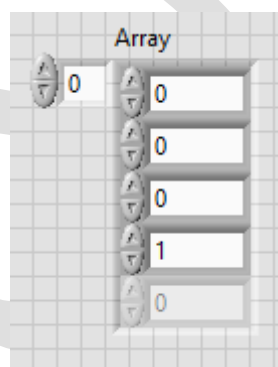
شکل ۲۴ تغییر اندازه‌ی آرایه

با حرکت دادن نقاط آبی، سائز آرایه نیز تغییر می‌کند. برای ایجاد یک آرایه‌ی یک بعدی، می‌توان آنرا به شکل یک بردار ستونی تغییر شکل داد:



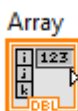
شکل ۲۵ ایجاد آرایه یک بعدی با ۵ درایه

توجه نمایید تا زمانی که عددی به درایه‌ی آرایه اختصاص داده نشده است، آن درایه راه اندازی نشده است. یعنی اگر در ماتریس ۵ در ۱ بالا، تنها ۴ درایه اول را عدد بدهیم، این آرایه مانند یک ماتریس ۴ در ۱ عمل خواهد کرد:



شکل ۲۶ راه‌اندازی درایه‌های یک آرایه

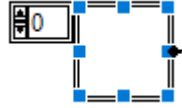
با اختصاص عدد ۱ به درایه چهارم این آرایه، تمامی درایه‌های پیش از آن، مقدار پیش‌فرض صفر را اختیار خواهند کرد و درایه‌های بعد از آن راه اندازی نخواهند شد. همانطور در هنگام ایجاد متغیرهای عددی یا باینری، در فضای Block Diagram ترمینال ایجاد می‌گردید در اینجا نیز برای این آرایه در فضای دیاگرام بلوکی نیز ترمینال ایجاد شده است که از آن می‌توان در روند برنامه‌نویسی استفاده کرد:



شکل ۲۷ ترمینال ایجاد شده برای آرایه در Block Diagram

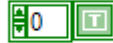
آرایه ثابت

برای ایجاد آرایه ثابت، طبیعتاً چون کاربر دخالتی در ایجاد یا تغییر این ثوابت ندارد، از محیط Block Diagram اقدام کرده و با استفاده از پالت Functions، به بخش Array رفته و Array Constant را انتخاب می‌نماییم:



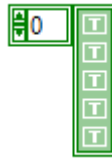
شکل ۲۸ ایجاد یک آرایه ثابت

در اینجا نیز مانند قبل، برای تشکیل آرایه، باید نوع ثابت آنرا مشخص نماییم. برای مثال می‌خواهیم این آرایه دارای ثوابت باینری باشد. برای این منظور از پالت Functions به قسمت Boolean رفته و گزینه‌ی True Constant را انتخاب کرده و آنرا درون کادر آرایه قرار می‌دهیم:



شکل ۲۹ تشکیل آرایه ثابت از نوع باینری

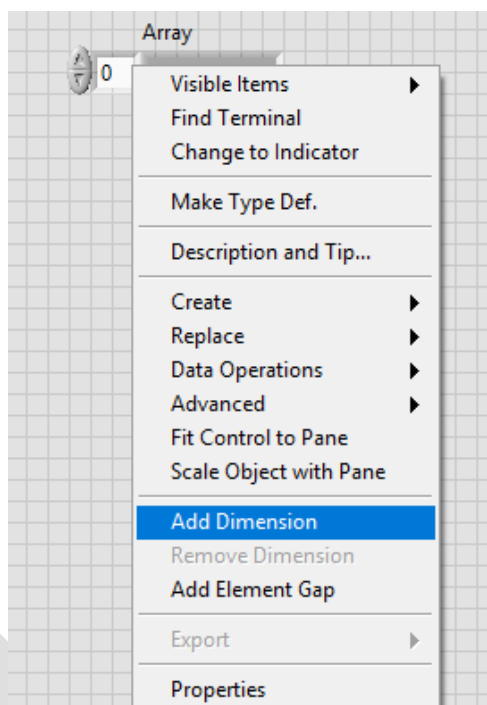
مانند قبل می‌توان این آرایه را به صورت یک بردار تعریف کرد که دارای چندین ثابت می‌باشد:



شکل ۳۰ تغییر اندازه آرایه ثابت باینری

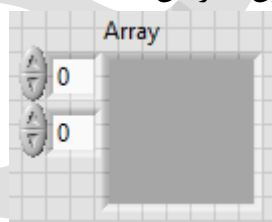
ساخت آرایه‌ی چند بعدی

برای ساخت آرایه‌ی چند بعدی یا همان ماتریس، مانند روندی که در ساخت آرایه‌ی یک بعدی عمل کردیم، عمل می‌نماییم. برای مثال، برای ساخت ماتریس دو بعدی، مانند قبل در محیط Front Panel، یک آرایه را قرار داده و سپس با راست کلیک بر روی کنترل آرایه (عدد ورودی در بالا و سمت چپ بلوک) و انتخاب گزینه Add Dimension، یک بعد دیگر به آرایه اضافه می‌نماییم:



شکل ۳۱ اضافه کردن بعد به آرایه

در این صورت، یک کنترل آرایه دیگر در زیر کنترل قبلی اضافه می‌شود. کنترل بالایی بیانگر اندیس ردیف ماتریس و کنترل پایینی بیانگر اندیس ستون ماتریس است.



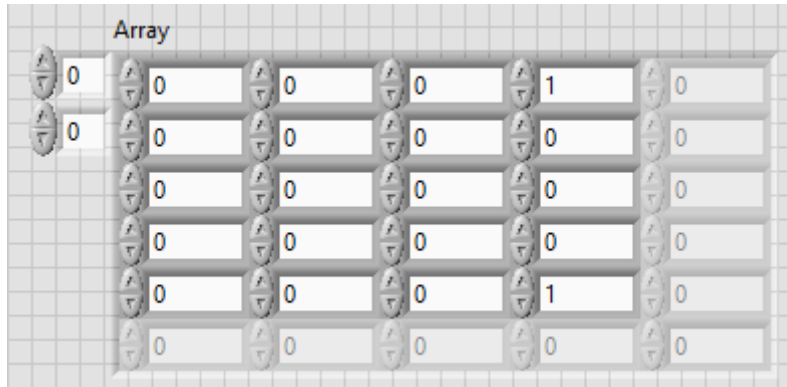
شکل ۳۲ یک آرایه با دو بعد

توابع آرایه‌ها

همانطور که می‌دانیم آرایه‌ها مانند متغیرهایی هستند که تعداد زیادی از داده‌ها با جنس یکسان را در خود ذخیره کرده‌اند. در اغلب موارد لازم است تا بر روی داده‌های موجود، عملیات و پردازشی انجام شود تا نتیجه‌ی مطلوب بدست آید یا به گونه‌ای از این داده‌ها در روند برنامه‌ی خود استفاده نماییم. مانند متغیرهای عددی یا باینری، برای آرایه‌ها نیز توابعی تعریف شده است که در اینجا به معرفی آنها می‌پردازیم.

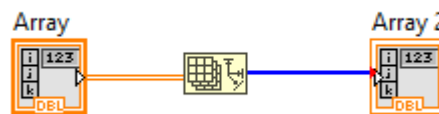
ابعاد آرایه

یکی از مهمترین و ابتدایی‌ترین عملی که در آرایه‌ها با آن سروکار خواهیم داشت بدست آوردن ابعاد یک آرایه است. برای این منظور در نرم‌افزار LabVIEW، بلوکی تعبیه شده است که با اتصال آرایه‌ی مورد نظر، بسته به ابعاد آن، یک عدد یا آرایه یک بعدی را که به منظور تعداد درایه در هر بعد آن است به دست می‌دهد. برای مثال ماتریس دو بعدی زیر را در نظر بگیرید که دارای ابعاد ۵ در ۴ می‌باشد (۵ ردیف و ۴ ستون):

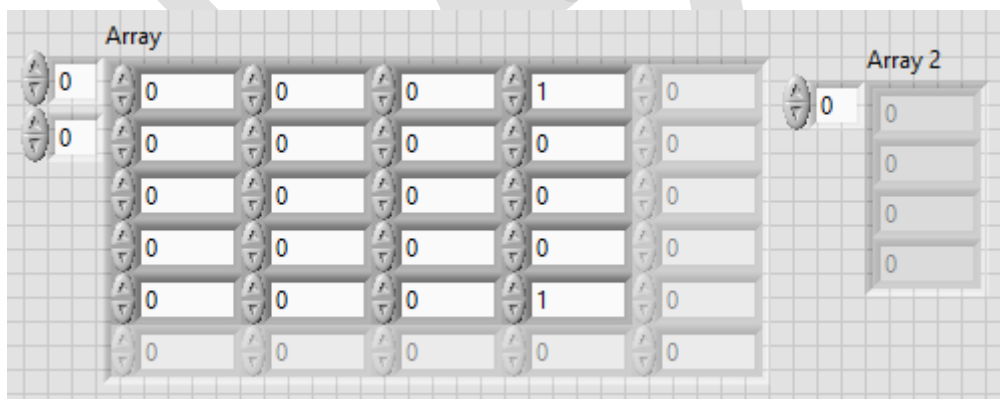


شکل ۳۳ ایجاد یک آرایه با دو بعد

برای بدست آوردن ابعاد این ماتریس در محیط Block Diagram، از پالت Functions و قسمت Array، بلوک Array Size را انتخاب کرده و قرار می‌دهیم. با اتصال ماتریس بالا به گرهی ورودی این بلوک و همچنین اتصال یک آرایه یک بعدی با حداقل دو درایه (زیرا قرار است هر کدام از درایه‌ها تعداد درایه در هر بعد ماتریس ورودی را نشان دهند) برنامه‌ای به شکل زیر داریم:

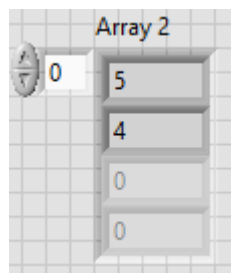


شکل ۳۴ اتصال بلوک اندازه‌ی آرایه به آرایه‌ی ایجاد شده



شکل ۳۵ نمایش اندازه آرایه در بلوک آرایه ۲

با اجرای برنامه همانطور که انتظار می‌رود، در درایه‌ی اول ماتریس ابعاد مقدار ۵ و در درایه‌ی دوم آن مقدار ۴ نمایان می‌گردد:



شکل ۳۶ نمایش اندازه آرایه

همانطور که مشاهده می‌کنید، در ماتریس ابعاد تنها از ۲ درایه‌ی اول استفاده شده است و سایر درایه‌ها تعریف و راه‌اندازی نشده‌اند.

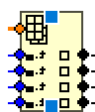
فراخوانی درایه

گاهی لازم است تا به طور خاص به یک یا چند درایه‌ی خاص دسترسی داشته باشیم، برای این منظور با داشتن اندیس درایه یا درایه‌های مورد نظر می‌توان آنها را فراخوانی کرد. بلوک Index Array که در بخش Array در پالت توابع می‌باشد، به همین منظور تعبیه شده است. با اتصال آرایه و همچنین شماره اندیس(ها) به این بلوک، می‌توان در خروجی درایه(ها)ی متناظر با آن اندیس(ها) را استخراج کرد.



شکل ۳۷ بلوک فراخوانی درایه

برای فراخوانی چند درایه تنها کافیست با کمک دو مربع آبی رنگ در بالا و پایین بلوک، اندازه‌ی آنرا تغییر دهید تا تعداد گره‌های اندیس-درایه اضافه شود:



شکل ۳۸ تغییر اندازه بلوک فراخوانی درایه

جایگزین کردن آرایه

از این بلوک برای جایگزین کردن یک اندیس یا ردیف یا ستون در یک آرایه استفاده می‌شود. برای این منظور لازم تا به گره‌های ورودی این بلوک آرایه‌ی اولیه، شماره‌ی اندیس و درایه یا آرایه‌ی جایگزین را متصل کنید تا در خروجی آرایه با درایه یا آرایه‌ی جایگزین شده را دریافت کنید.



شکل ۳۹ بلوک جایگزین کردن درایه

در این بلوک نیز می‌توان بسته به نیاز خود سایز آنرا برای افزایش تعداد اندیس-درایه تغییر دهید.

سایر توابع

در ادامه به معرفی مختصر سایر توابع مرتبط و کاربردی برای آرایه‌ها می‌پردازیم:

- **قرار دادن در آرایه:** یک درایه یا آرایه را در اندیس مشخص شده به یک آرایه n بعدی اضافه می‌کند.



شکل ۴۰ بلوک قرار دادن در آرایه

- **حذف کردن از آرایه:** یک درایه یا آرایه را از اندیس مشخص شده حذف کرده و در خروجی آرایه‌ی اصلاح شده و همچنین درایه یا آرایه حذف شده داده می‌شود.



شکل ۴۱ بلوک حذف از آرایه

- راه اندازی آرایه: یک آرایه n بعدی با درایه‌ی داده شده ایجاد می‌شود.



شکل ۴۲ بلوک راه‌اندازی آرایه

- ساخت آرایه: چند آرایه را به یکدیگر پیوند می‌دهد یا یک یا چند درایه را به آرایه اولیه اضافه می‌کند.



شکل ۴۳ بلوک ساخت آرایه

- زیرمجموعه‌ی آرایه: بخشی از آرایه را از اندیس مشخص شده به طول داده شده، استخراج می‌کند.



شکل ۴۴ بلوک زیرمجموعه آرایه

- ماکزیمم و مینیمم: مقدار ماکزیمم و مینیمم موجود در آرایه و همچنین اندیس آنها را می‌دهد.

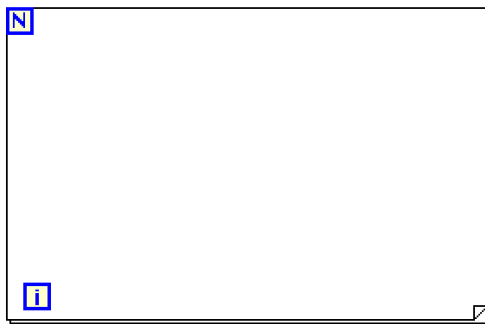


شکل ۴۵ بلوک ماکزیمم و مینیمم

ساختارها و حلقه‌ها

حلقه For

حلقه‌ی For به ازای تعداد دفعات مشخصی، یک پروسه را تکرار می‌نماید. برای دسترسی به این حلقه در محیط Block Diagram به بخش Structures رفته و For Loop را انتخاب می‌نماییم:



شکل ۴۶ حلقه For در محیط Block Diagram

هر برنامه‌ای که داخل این حلقه طراحی شود برای N بار تکرار خواهد شد و سپس برنامه از حلقه خارج می‌شود. برای مشخص کردن دفعات تکرار این حلقه، کفایت گرهی بلوک N را به یک مقدار عددی متصل نماییم. این مقدار عددی می‌تواند یک عدد ثابت، کنترل عددی یا از هر طریق دیگری بدست آید. همچنین بلوک i در داخل این حلقه همان شمارشگر حلقه یا iteration است که شماره‌ی تکرار حلقه را نشان می‌دهد که می‌توان از گرهی خروجی آن در برنامه استفاده کرد. این مقدار همیشه از 0 شروع می‌شود، یعنی در دفعه‌ی اول مقدار i برابر صفر است.

در واقع حلقه‌ی For به ازای شرط $N > i$ برقرار است و تکرار می‌شود و در هر بار تکرار مقدار N یک واحد افزایش می‌یابد. بنابراین در صورتی که مقدار N را صفر یا عددی منفی قرار دهیم حلقه‌ی For هرگز اجرا نمی‌شود. همچنین متغیرهای i و N از نوع Integer بوده و در صورت اعمال یک مقدار Float به این متغیرها، به صورت خودکار به نزدیکترین عدد Integer رند می‌شوند.

حلقه‌ی For با بلوک شرط توقف

قابلیت مهمی که در حلقه‌ی For وجود دارد این است که می‌توان علاوه بر تکرار حلقه به دفعات مشخص، به ازای شرط مشخصی نیز از حلقه خارج شد. با راست کلیک بر روی حلقه‌ی For و انتخاب Conditional Terminal می‌توان یک بلوک شرط برای حلقه‌ی For ایجاد کرد که در صورت ارضا شدن این شرط نیز، برنامه از حلقه خارج می‌شود:



شکل ۴۷ حلقه‌ی For با بلوک شرط توقف

شرط ایجاد شده مانند شرط حلقه‌ی While که در ادامه خواهیم گفت عمل می‌کند.

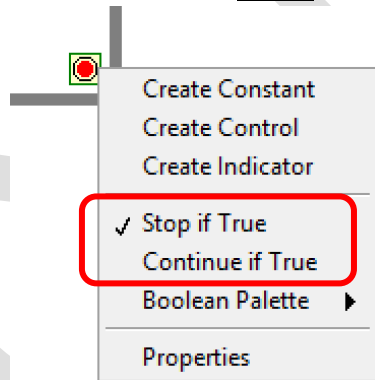
حلقه While

حلقه‌ی While تا هنگام ارضای شرط تعیین شده تکرار خواهد شد. ساختار این حلقه به شکل زیر می‌باشد:



شکل ۴۸ حلقه While در Block Diagram

بلوک i مانند حلقه‌ی For تعداد دفعات تکرار برنامه را می‌دهد. بلوک شرط به طور پیش فرض در سمت راست و پایین حلقه قرار می‌گیرد. این بلوک دارای یک گره‌ی ورودی است که باید به یک متغیر باینری متصل گردد. بلوک شرط می‌تواند به دو صورت شرط را تلقی کند: (۱) توقف حلقه به ازای True بودن گره‌ی ورودی؛ (۲) ادامه‌ی حلقه به ازای True بودن گره‌ی ورودی.

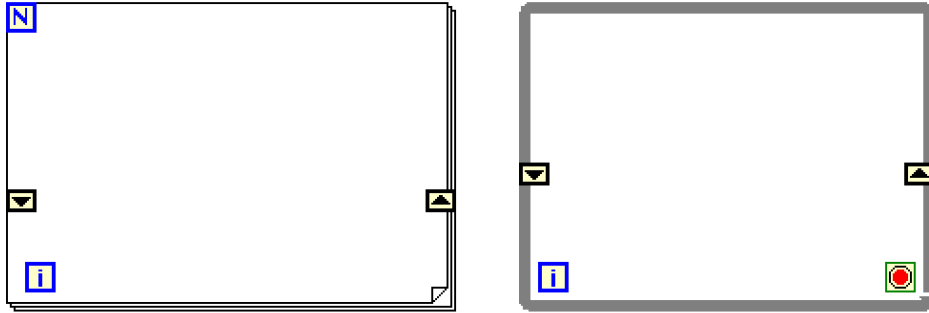


شکل ۴۹ تغییر وضعیت شرط توقف

به طور پیش فرض بلوک شرط، حالت اول را در نظر می‌گیرد. در این حالت برای ایجاد یک حلقه‌ی بینهایت (تکرار دائمی حلقه) کفایت یک مقدار ثابت صفر یا False را به بلوک شرط متصل نماییم. با راست کلیک بر روی بلوک شرط و انتخاب Create Constant می‌توان یک مقدار ثابت صفر را به این بلوک متصل کرد. در این حالت برنامه با ورود به حلقه‌ی While، هیچگاه از آن خارج نخواهد شد.

شیفت رجیستر (Shift Register)

برای انتقال مقادیر بدست آمده در هر بار تکرار حلقه (حلقه For و While) به تکرارهای آینده، می‌توان از شیفت رجیستر استفاده کرد. با راست کلیک بر روی حلقه و انتخاب Add Shift Register، همانند شکل زیر در طرفین حلقه دو بلوک ایجاد می‌شود:

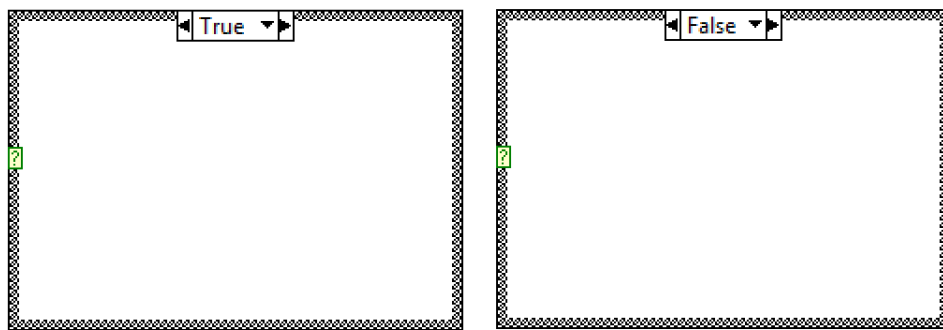


شکل ۵۰ ایجاد شیفت رجیستر در حلقه While و For

بلوک سمت راست با فلش رو به بالا مشخص شده است که مقادیر را در هر تکرار ذخیره می‌کند، در این حالت نرم‌افزار به طور خودکار این مقدار را به تکرار بعدی منتقل می‌کند. شیفت رجیستر قادر است تا هر نوع داده را دریافت کرده و البته هر دو داده‌ی متصل به دو بلوک شیفت رجیستر باید از یک نوع باشند. *تعداد شیفت رجیسترها محدود نبوده و می‌توان آنها را به هر تعدادی که نیاز است به حلقه اضافه کرد. همچنین می‌توان با تغییر اندازه‌ی بلوک سمت چپ شیفت رجیستر، به مقادیر در تکرارهای قبل تر دسترسی داشت.

ساختار Case Structure

این ساختار دارای دو یا چند مورد (Case) است که در صورت برآورده شدن شرط هر یک موردها، برنامه‌ی آن بخش اجرا می‌شود. این ساختار در هر لحظه تنها یک مورد را اجرا می‌کند. مقداری که به ورودی این ساختار داده می‌شود تعیین می‌کند که کدام شرط باید اجرا شود.



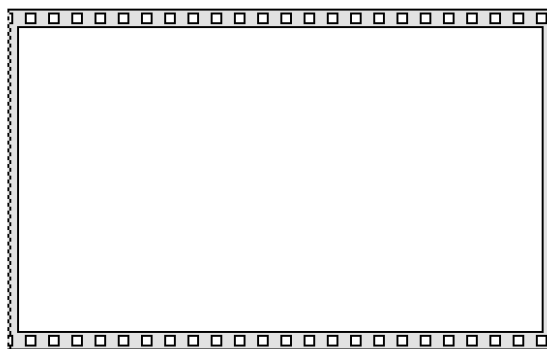
شکل ۵۱ دو مورد مختلف در ساختار Case Structure

بلوکی که در سمت چپ دیواره‌ی این ساختار وجود دارد و با علامت «؟» مشخص شده است شرط این ساختار می‌باشد. در صورتی که این مقدار با هریک از موردها تطابق داشته باشد برنامه‌ی مربوط به آن مورد اجرا خواهد شد. به صورت پیش فرض برای این ساختار دو مورد یک (True) و صفر (False) در نظر گرفته شده است اما می‌توان این ساختار را برای انواع دیگر داده‌ها نیز استفاده نمود. موردهای این ساختار در منوی کشویی بالای آن قرار گرفته‌اند. با انتخاب هر مورد می‌توانید مقدار آنرا تغییر دهید.

در صورتی که ورودی داده شده به این ساختار باینری نباشد باید یک مورد به صورت پیش فرض تعیین شود تا در صورت عدم تطابق سایر موردها، برنامه‌ی این مورد اجرا گردد. پیش فرض کردن هر مورد، با انتخاب آن مورد و راست کلیک بر روی آن و انتخاب گزینه‌ی **Make This The Default Case** انجام می‌پذیرد.

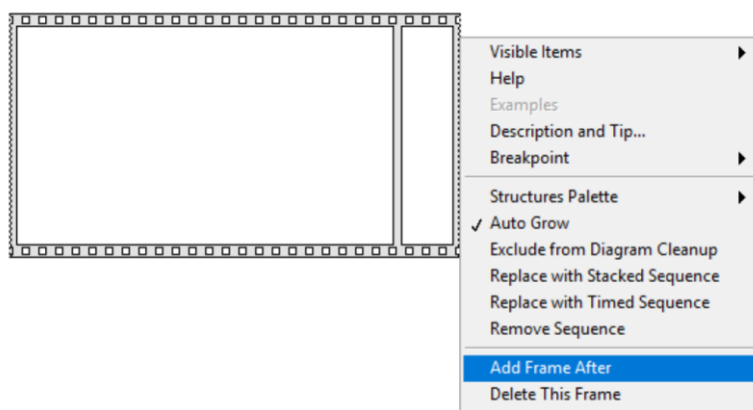
ساختار Flat Sequence

این ساختار برای ایجاد ترتیب در روند اجرای بخش یا بخش‌هایی از برنامه استفاده می‌شود. همانطور که گفته شد نرم‌افزار LabVIEW لزوماً از چپ به راست یا بالعکس اجرا نمی‌شود و هر زمانی که مقادیر متصل به گره‌ی هر بلوک در دسترس باشد برنامه اجرا خواهد شد.



شکل ۵۲ ساختار Flat Sequence

به هر بخش از ساختار Flat Sequence، فریم گفته می‌شود که هر ساختار Flat Sequence می‌تواند شامل یک یا چند فریم باشد. از آنجایی که این ساختار از چپ به راست و زمانی که تمام داده‌های مرتبط با یک فریم در دسترس باشد، اجرا می‌شود، در نتیجه با استفاده از این ساختار می‌توان روند مورد نظر اجرای برنامه را به طور دقیق پیاده نمود. برای اضافه کردن فریم به این ساختار، بر روی آن راست کلیک کرده و بر روی گزینه‌ی **Add Case Before** یا **Add Case After** کلیک نمایید.



شکل ۵۳ ایجاد فریم جدید در ساختار Flat Sequence

داده‌ها پس از اجرای کامل هر فریم، می‌توانند از آن خارج شوند، در نتیجه ممکن است ورودی یک فریم وابسته به خروجی سایر فریم‌های پیش از خود باشد.
* تا زمانی که عملیات آخرین فریم به اتمام نرسد هیچ داده‌ای از این ساختار خارج نمی‌گردد.

تمرین فصل دوم

۱. برنامه‌ای طراحی کنید تا دما را به سلسیوس از کاربر دریافت کرده و آنرا به فارنهایت نمایش دهد (برای تبدیل سلسیوس به فارنهایت از رابطه $^{\circ}F = 1.8 \times ^{\circ}C + 32$ استفاده کنید).



۲. با استفاده از حلقه For و بدون استفاده از بلوک ضرب (Multiply)، برنامه‌ای طراحی کنید تا دو عدد را از کاربر گرفته و حاصل ضرب آنها را نمایش دهد.



۳. با استفاده از حلقه‌ی For، یک تایمر بسازید تا با شروع برنامه زمان سپری شده را نمایش دهد.



۴. یک ماشین حساب ساده بسازید که دو عدد را از کاربر گرفته و با تعیین یکی از چهار عملگر (جمع، تفریق، ضرب و تقسیم) این عملیات را بر روی آن دو عدد انجام داده و نتیجه را نمایش دهد.



فصل سوم

«برنامه نویسی MINDSTORMS»

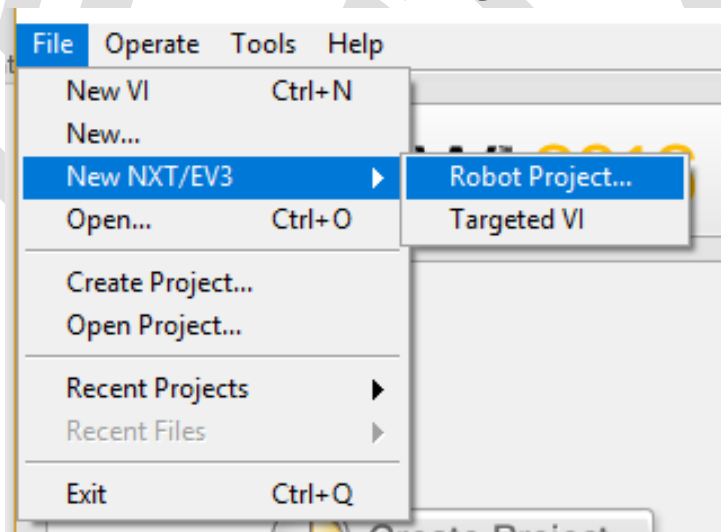
مقدمه‌ای بر ماژول MINDSTORMS

همانطور که می‌دانید نرم‌افزار LabVIEW برای متصل شدن به سخت افزارهای گوناگون طراحی شده است. به همین منظور برای اتصال این نرم‌افزار به سخت‌افزارهای پشتیبانی شده، لازم است تا درایور آن نصب گردد. ماژول MINDSTORMS که توسط همین شرکت ارائه شده است برای اتصال نرم‌افزار LabVIEW به پکیج Lego MINDSTORMS استفاده می‌شود.

با استفاده از این ماژول می‌توان VI طراحی شده را به دو صورت پیاده‌سازی کرد: (۱) اجرای VI بر روی کامپیوتر و تبادل فرمان و اطلاعات با بریک EV3/NXT، (۲) اجرای VI به طور مستقل بر روی بریک EV3/NXT. همچنین با کمک این ماژول می‌توان موتورها و سرووهای TETRIX متصل به بریک EV3/NXT را کنترل کرد.

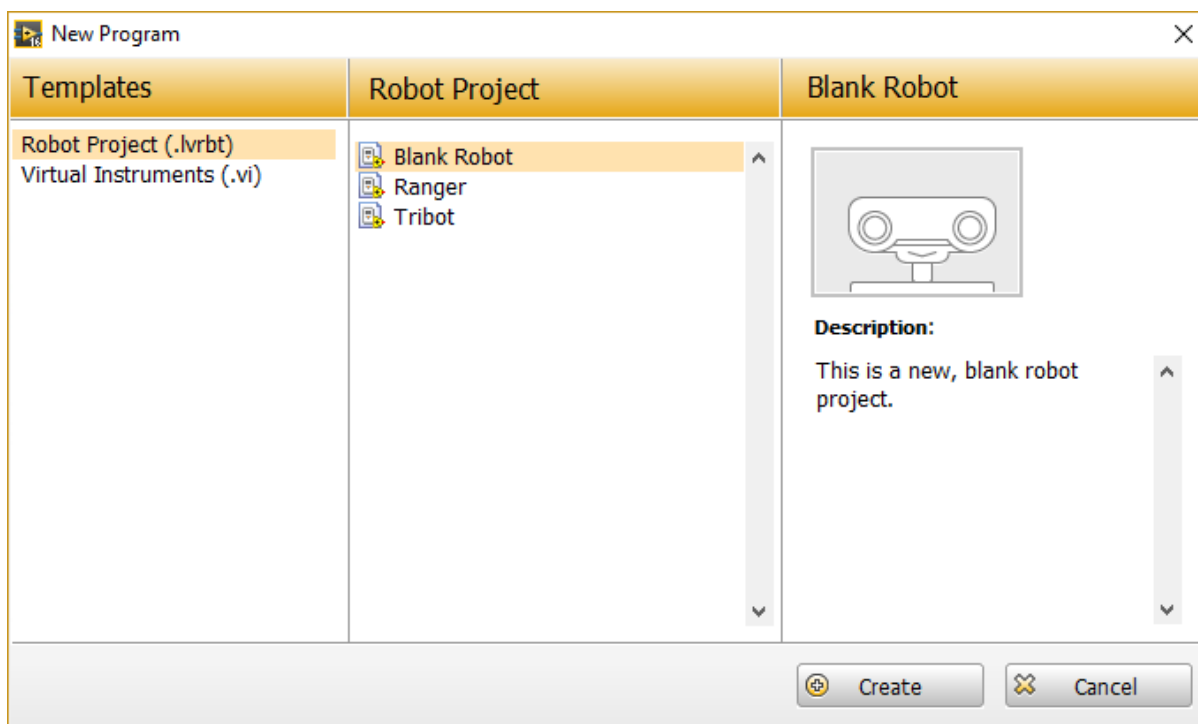
در این قسمت به هر دو صورت یک VI را پیاده‌سازی خواهیم کرد.

برای شروع کار با این ماژول، از صفحه‌ی اصلی برنامه و نوار منو، به بخش File رفته و گزینه‌ی New Robot Project → NXT/EV3 را انتخاب می‌نماییم.



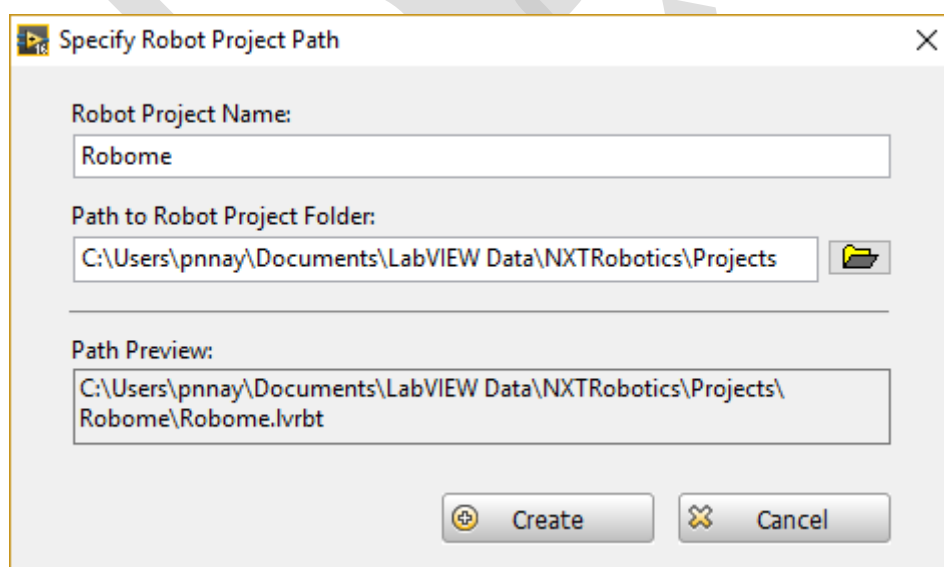
شکل ۱ ایجاد پروژه‌ی جدید EV3

با انتخاب این گزینه، پنجره‌ی زیر باز می‌گردد:



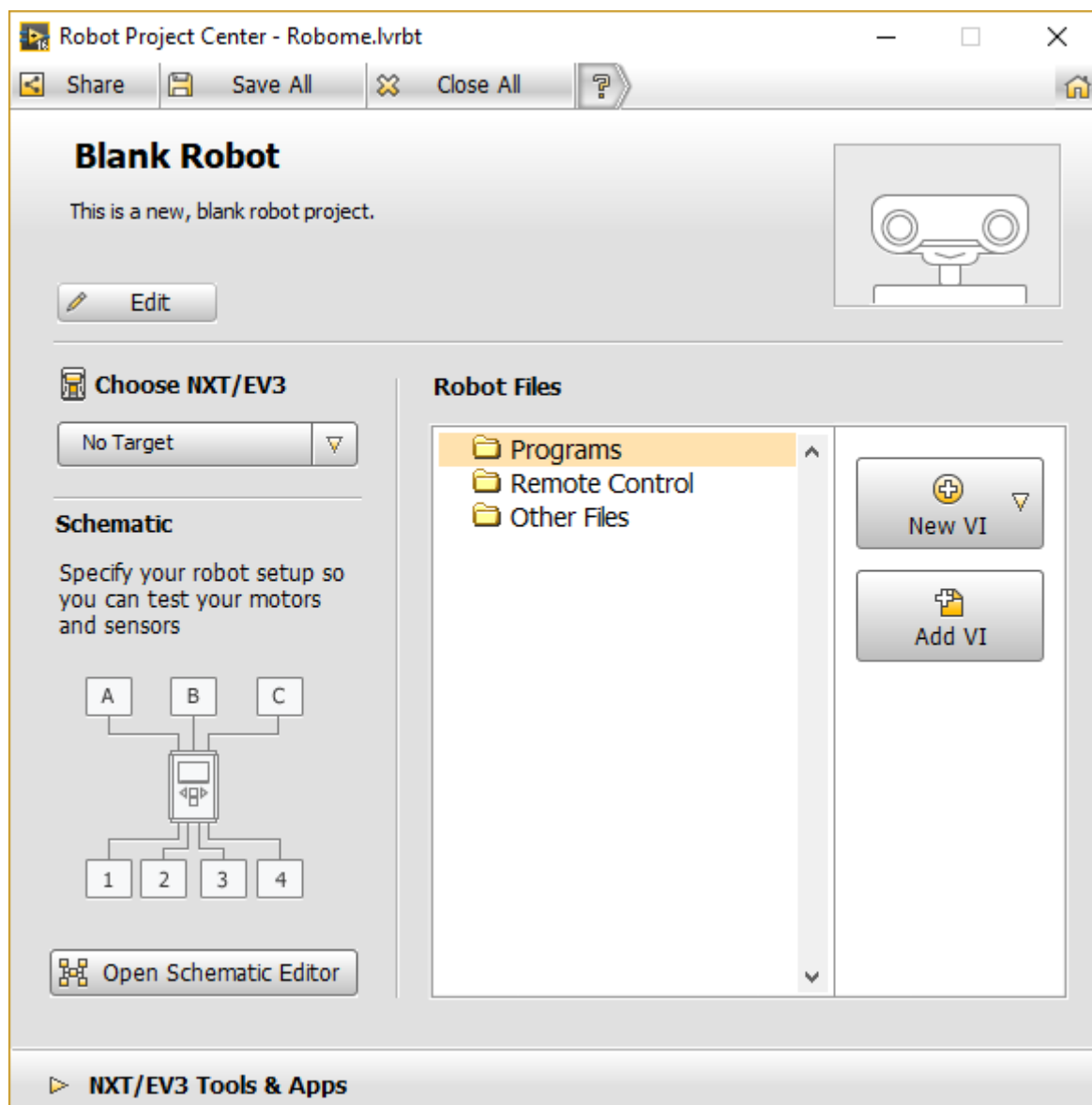
شکل ۲ پنجره‌ی ایجاد پروژه‌ی EV3

بر روی دکمه Create کلیک کرده و برای پروژه‌ی خود نامی را به دلخواه انتخاب کنید و سپس دکمه‌ی Create را کلیک کنید:



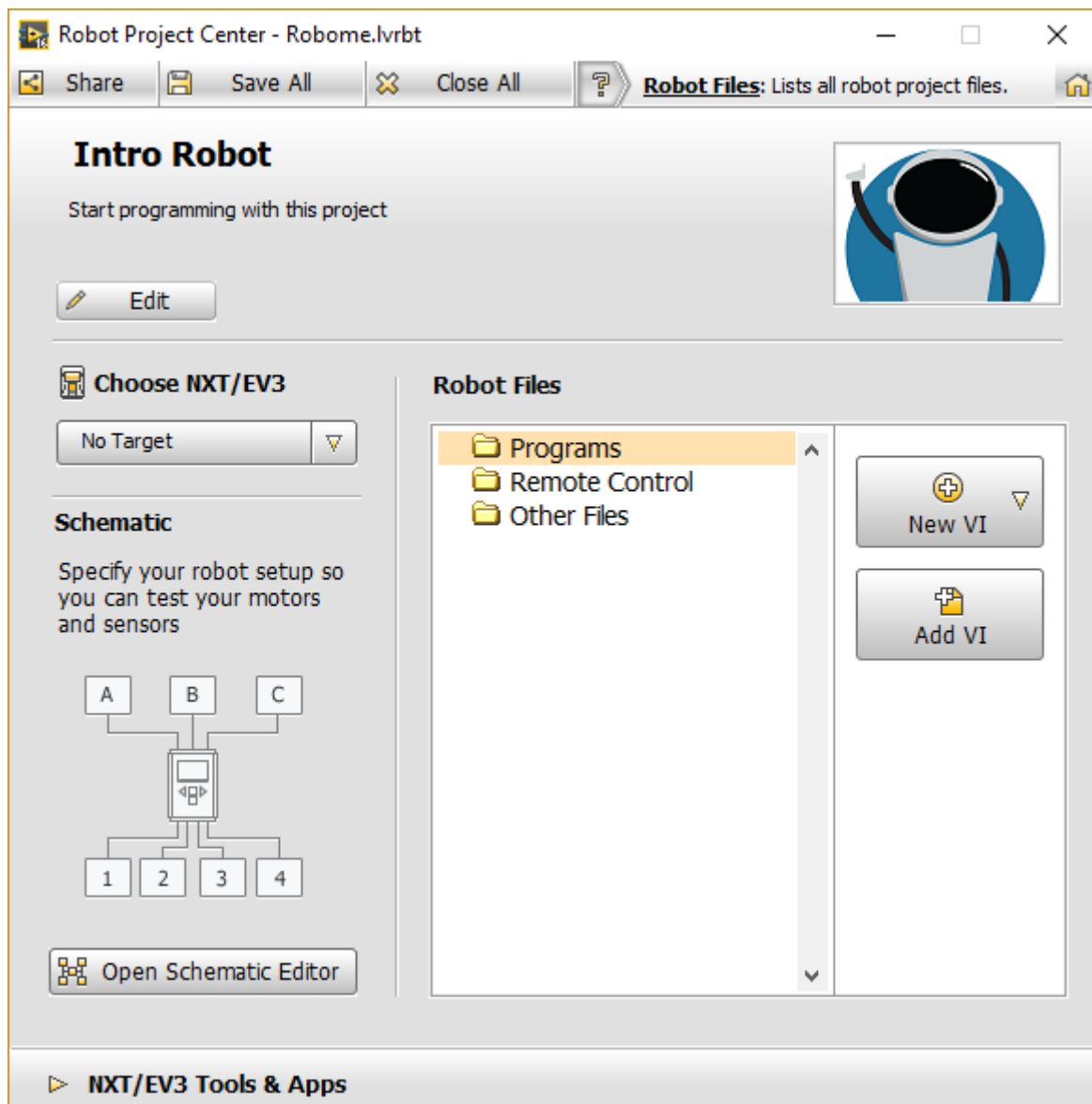
شکل ۳ مسیر ایجاد پروژه‌ی جدید

با اینکار پروژه‌ی رباتیک جدیدی تعریف می‌گردد که از طریق پنجره‌ی زیر می‌توان آنرا کنترل نمود:



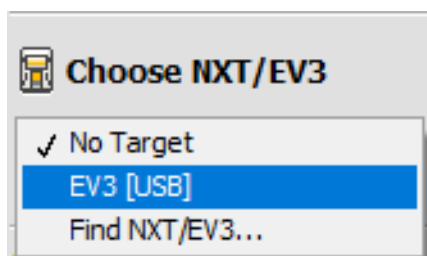
شکل ۴ پنجره‌ی کنترل پروژه

در قسمت بالای این پنجره نام، توضیحات و تصویر مربوط به ربات وجود دارد، با کلیک بر روی دکمه Edit می‌توان آنرا ویرایش نمود.



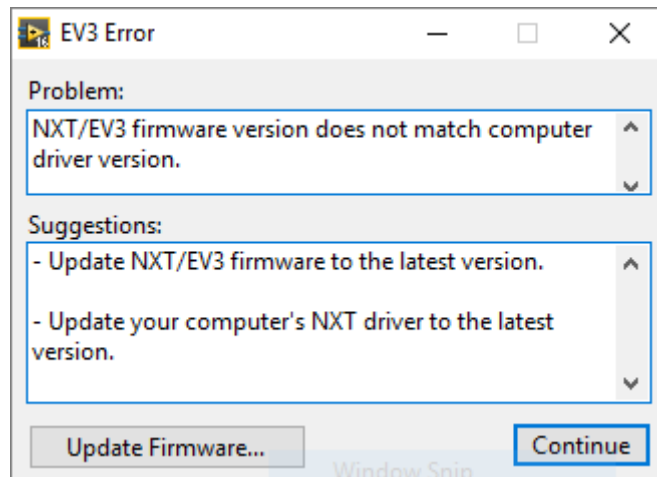
شکل ۵ وارد کردن نام، توضیحات و تصویر ربات

در قسمت Choose NXT/EV3، بریک مورد نظر خود را که به کامپیوتر متصل شده است انتخاب می‌نمایید. برای این منظور بریک خود را با کابل USB به کامپیوتر متصل نمایید و بعد از شناسایی شدن بریک، آنرا از منوی کشویی موجود در این قسمت انتخاب کنید:



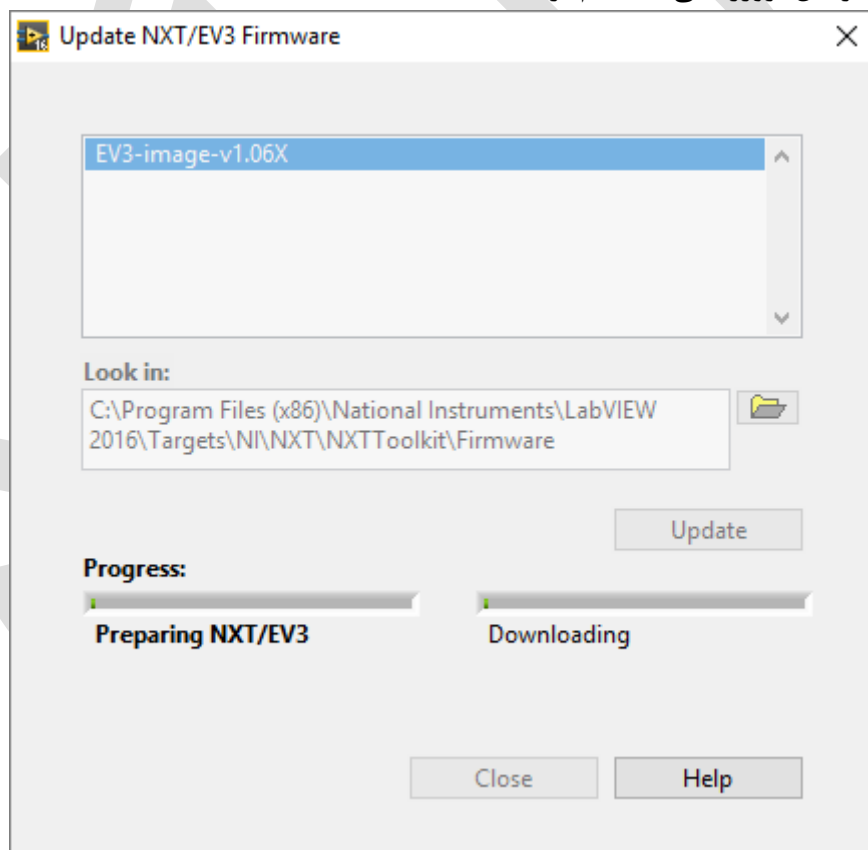
شکل ۶ انتخاب EV3 متصل به کامپیوتر

* در صورتی که بعد از انتخاب بریک، با پیغام EV3 error مواجه شدید بر روی دکمه Update Firmware... کلیک کنید تا بریک شما متناسب با ورژن نرم‌افزار LabVIEW بروزرسانی شود:



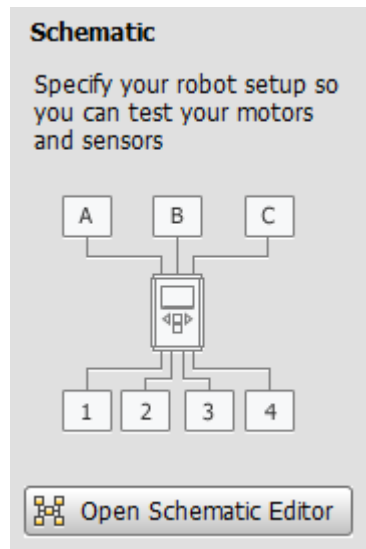
شکل ۷ بروزرسانی ورژن بریک

بعد از انتخاب این گزینه، در پنجره‌ی Update NXT/EV3 Firmware بر روی دکمه Update کلیک کرده و منتظر بمانید تا مراحل بروزرسانی به اتمام برسد:



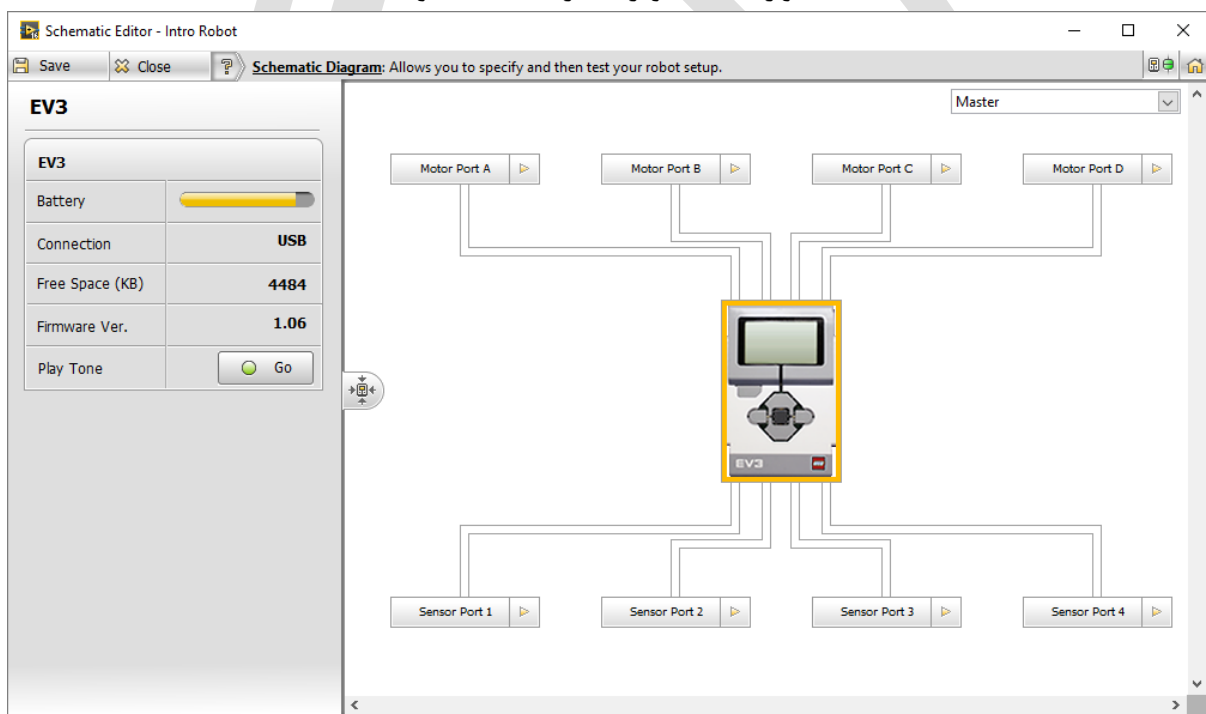
شکل ۸ پروسه‌ی بروزرسانی بریک

بعد از نمایش پیغام **Successfully Downloaded Firmware!** بر روی دکمه Close کلیک نمایید. بخش دیگری که در پنجره‌ی Robot Project Center وجود دارد Schematic یا همان شماتیک بریک می‌باشد.



شکل ۹ شماتیک بریک در صفحه کنترل پروژه

با انتخاب Open Schematic Editor وارد محیط ویرایشگر شماتیک خواهید شد:

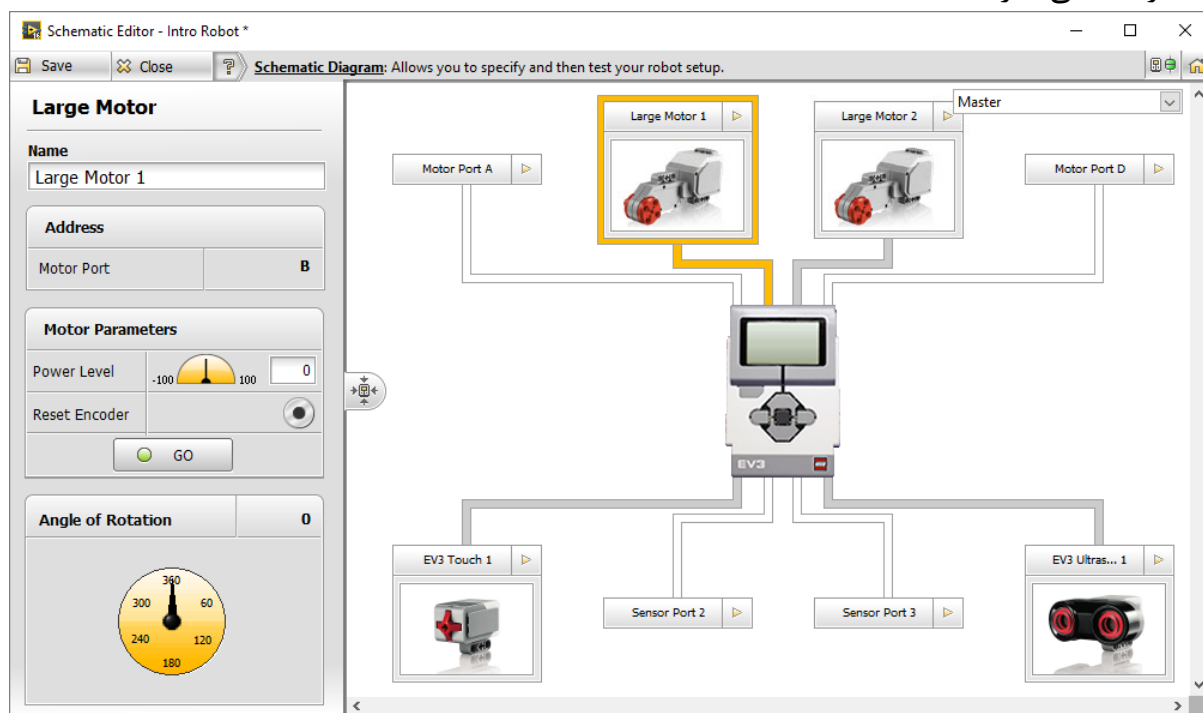


شکل ۱۰ پنجره‌ی ویرایش شماتیک بریک

همانطور که در سمت چپ این محیط مشاهده می‌نمایید، اطلاعات کلی بریک اعم از شارژ باتری، نوع اتصال، فضای خالی، ورژن فریم‌ور نمایش داده می‌شود. همچنین در صورتی که به طور همزمان به چند بریک متصل هستید، می‌توانید با کلیک بر روی دکمه Go مقابل عبارت Play Tone با پخش صدای بوق، بریک مورد نظر را شناسایی نمایید.

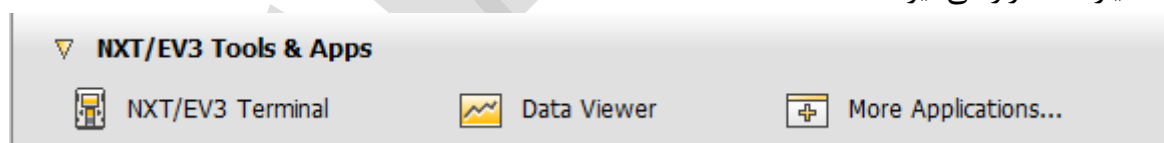
در فضای راست این پنجره شماتیکی از بریک و پورت‌های آنرا مشاهده می‌نمایید. با کلیک بر روی فلش کوچک کنار هر پورت، می‌توانید موتور یا سنسور متصل به آن پورت را انتخاب نمایید تا در آینده و در حین طراحی روند برنامه از این چیدمان کمک بگیرید. همچنین برای انجام تست‌های مختلف بر روی موتورها یا سنسورهای

متصل به بریک، با انتخاب هر یک از آنها، پنلی در سمت چپ نمایش داده می‌شود که امکانات متنوعی را در اختیار شما می‌گذارد.



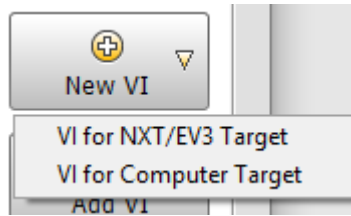
شکل ۱۱ نمایش موتورها و سنسورهای متصل به بریک

در قسمت Robot Files در پنجره Robot Project Center، فایل VI ها و سایر پروژه‌ها نمایش داده می‌شوند. در پایین پنجره Robot Project Center، با کلیک بر روی NXT/EV3 Tools & Apps برنامه‌ها و ابزار جانبی در اختیار شما قرار می‌گیرد:



شکل ۱۲ نمایش سایر ابزارهای جانبی

ابزار NXT/EV3 Terminal پنجره‌ای را باز می‌نماید که در آن اطلاعات بریک را در اختیار شما قرار می‌دهد. همچنین می‌توانید از این ابزار برای تغییر نام بریک و ارسال، دریافت و ویرایش فایل از بریک استفاده نمایید. ابزار Data Viewer دریافت شده از سنسورهای متصل به بریک را به شما نمایش می‌دهد. گزینه More Applications... برنامه‌های جانبی دیگر را در اختیار شما قرار می‌دهد. حال که تمامی بخش‌های Robot Project Center را بررسی کرده‌ایم، آماده‌ی شروع برنامه‌نویسی برای بریک هستیم. برای ساخت یک VI و شروع برنامه‌نویسی، بر روی کلید New VI در بخش Robot Files کلیک کنید:

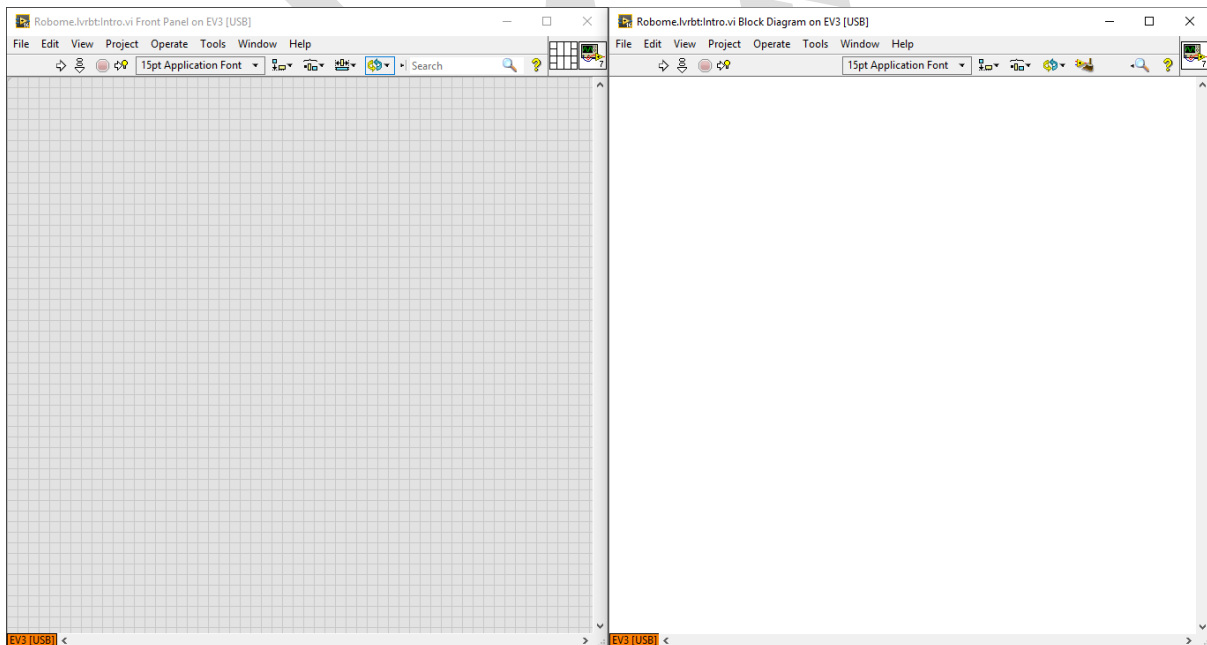


شکل ۱۳ ساخت برنامه متناسب با مقصد موردنظر

با کلیک بر روی این دکمه دو گزینه ظاهر می‌شود. در واقع شما می‌توانید مشخص کنید که برنامه‌ای که قصد دارید طراحی کنید به طور مستقل بر روی بریک اجرا شود یا بر روی کامپیوتر متصل به بریک اجرا شود. از گزینه‌ی اول یعنی VI for NXT/EV3 Target در مواقعی استفاده می‌کنیم که برنامه‌ی طراحی شده نیازی به کامپیوتر نداشته و به بریک به طور مستقل کنترل موتورها و سنسورها را در دست می‌گیرد. در این حالت مقصد VI بریک می‌باشد.

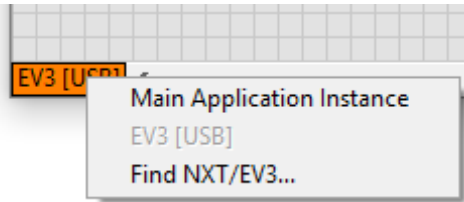
اما گزینه‌ی دوم یعنی VI for Computer Target برنامه‌ای طراحی می‌شود که توسط کامپیوتر اجرا می‌شود و کامپیوتر در صورت نیاز اطلاعاتی را از سنسورهای EV3 دریافت کرده یا فرامینی به موتورهای EV3 ارسال می‌نماید. از این نوع برنامه در موارد مختلفی می‌توان استفاده کرد برای مثال در مواردی که لازم است تا داده‌های خوانده شده در کامپیوتر نمایش داده شوند یا مواردی که کاربر متغیرهایی را در فضای Front Panel تغییر می‌دهد. در این حالت مقصد VI کامپیوتر می‌باشد.

با انتخاب هر یک از گزینه‌ها، بعد از انتخاب نام، یک VI جدید ایجاد شده و دو پنجره‌ی Front Panel و Block Diagram نمایش داده می‌شود:



شکل ۱۴ نمایش Front Panel و Block Diagram

با کمی دقت در پایین و سمت چپ هر پنجره یک فیلد وجود دارد که مقصد VI را مشخص می‌کند. با کلیک راست ماوس بر روی این فیلد، می‌توانید مقصد VI را تغییر دهید:



شکل ۱۵ انتخاب مقصد VI

گزینه‌ی Main Application Instance مقصد VI را به کامپیوتر تغییر می‌دهد. در صورتی که نرم‌افزار تاکنون بریک شما را شناسایی نکرده است می‌توانید با کلیک بر روی گزینه Find NXT/EV3 بریک خود را پیدا نمایید.

یکی از فرق‌های عمده‌ای که بین VI با مقصد بریک و کامپیوتر وجود دارد دکمه‌های اجرا یا آپلود VI است. در صورتی که مقصد بریک باشد دکمه‌های نوار ابزار VI به صورت اجرا (Run) و اعزام یا آپلود (Deploy) به نمایش درمی‌آیند اما در صورتی که مقصد کامپیوتر باشد دکمه‌ی آپلود به دکمه‌ی اجرای مداوم (Run Continuously) تغییر می‌کند.

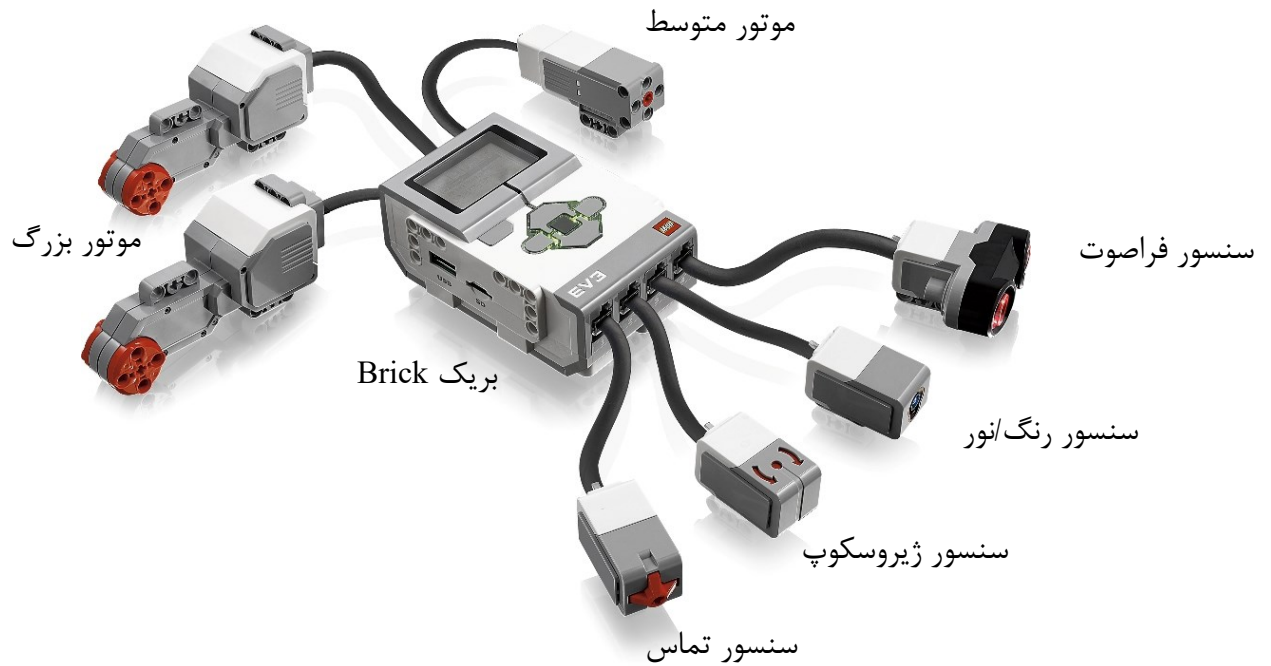
در واقع در حالتی که مقصد بریک می‌باشد، با کلیک بر روی دکمه اجرا، برنامه تنها یکبار اجرا شده و حتی بر روی حافظه‌ی بریک نیز ذخیره نخواهد شد، اما با کلیک بر روی دکمه آپلود یا اعزام، برنامه به بریک منتقل و در حافظه‌ی آن ذخیره شده و سپس می‌توان آنرا از طریق بریک اجرا نمود.

تفاوت دیگر این دو نوع برنامه، در پالت‌های ابزار و توابعی است که به طور پیش فرض در اختیار شما قرار می‌دهد. در صورتی که مقصد بریک باشد، پالت‌ها به طور پیش فرض بلوک‌های مرتبط با گروه MINDSTORMS را نمایش می‌دهند اما در صورتی که مقصد کامپیوتر باشد، پالت‌های اصلی برنامه نمایش داده می‌شوند، هرچند که در هر دو صورت با انتخاب نام گروه‌ها در هر دو پالت، می‌توان به تمامی گروه‌ها دسترسی داشت.

در قسمت بعد برنامه‌نویسی برای راه‌اندازی و کار با موتورها و سنسورها را مورد بررسی قرار خواهیم داد.

اجزای MINDSTORMS EV3

برای شروع برنامه‌نویسی EV3 ابتدا تجهیزات الکترونیکی از جمله موتورها و سنسورهای موجود در پکیج MINDSTORMS EV3 را بررسی می‌نماییم:



شکل ۱۶ اجزای مختلف پکیج MINDSTORM EV3

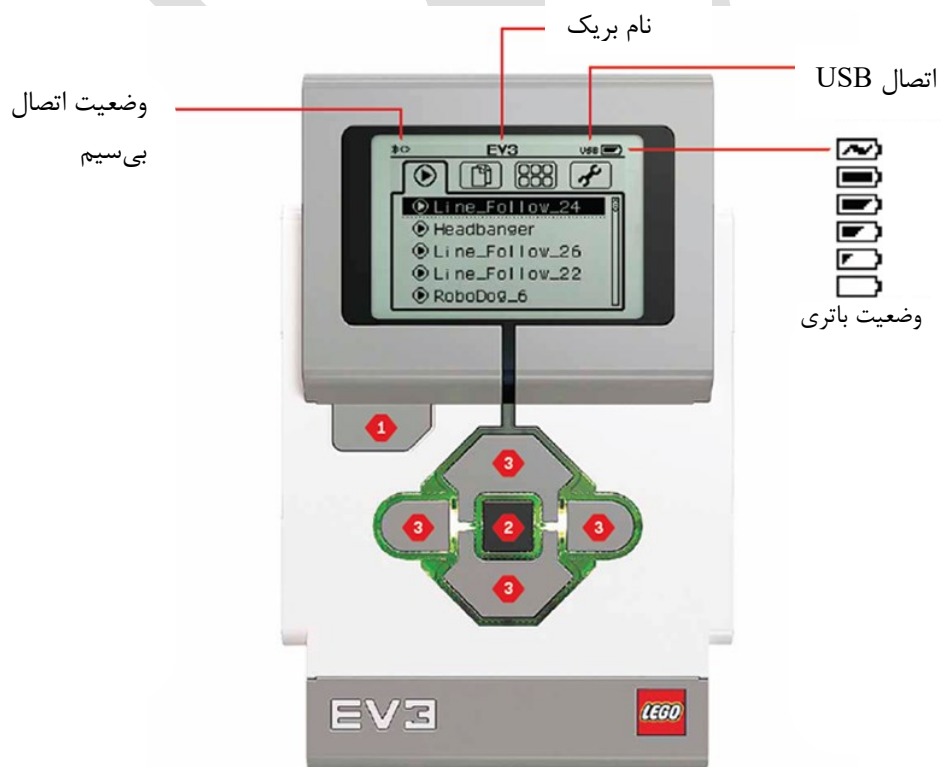
در ادامه به معرفی و توضیح هر یک از این اجزا می‌پردازیم.

بریک (Brick)



شکل ۱۷ بریک EV3

بریک در لغت به معنی بلوک یا آجر می‌باشد و در واقع مغز ربات‌های MINDSTROMS می‌باشند. برنامه‌های نوشته شده در بریک اجرا می‌شوند. اطلاعات مستخرج از سنسورها به بریک منتقل شده و دستورات مورد نظر توسط بریک به موتورها ارسال می‌شود. بر روی هر بریک ۵ کلید و ۸ پورت (درگاه) ورودی و خروجی وجود دارد. در شکل زیر بخش‌های مختلف بریک نمایش داده شده است:



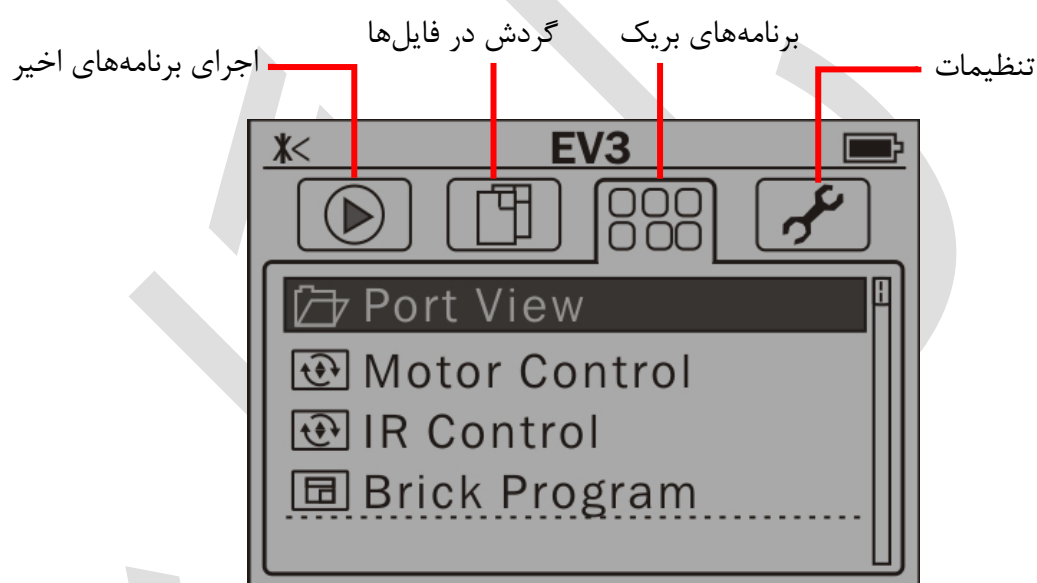
شکل ۱۸ بخش‌های مختلف بریک

۱. کلید بازگشت: از این کلید برای بازگشت به منوی قبل، خروج از برنامه‌ی در حال اجرا و خاموش کردن بریک استفاده می‌شود. برای خاموش کردن بریک لازم است تا به منوی اصلی بریک بازگشته و بعد از فشار دادن کلید بازگشت، گزینه‌ی ✓ را انتخاب کنید.

۲. کلید وسط: از این کلید برای انتخاب گزینه‌ی مورد نظر یا اجرای برنامه‌ها می‌توان استفاده کرد.

۳. کلیدهای جهت: این کلیدها چهار جهت اصلی (بالا، پایین، چپ و راست) می‌باشند. از این کلیدها برای گردش در منوهای بریک و همچنین اعمال ورودی به برنامه‌ها می‌توان استفاده کرد.

همانطور که می‌توان مشاهده کرد، بعد از روشن کردن بریک با نگه داشتن کلید وسط، چهار برگه یا منوی اصلی به نمایش درمی‌آید. برای انتخاب برگه‌ی مورد نظر از کلیدهای چپ و راست استفاده کنید.



شکل ۱۹ منوهای بریک

۱. منوی اجرای برنامه‌های اخیر (Run Recent): در این منو برنامه‌های اخیر که توسط بلوک اجرا شده به نمایش درمی‌آیند. همچنین برنامه‌های جدیدی که در داخل بلوک آپلود شده است نیز نمایش داده می‌شود.

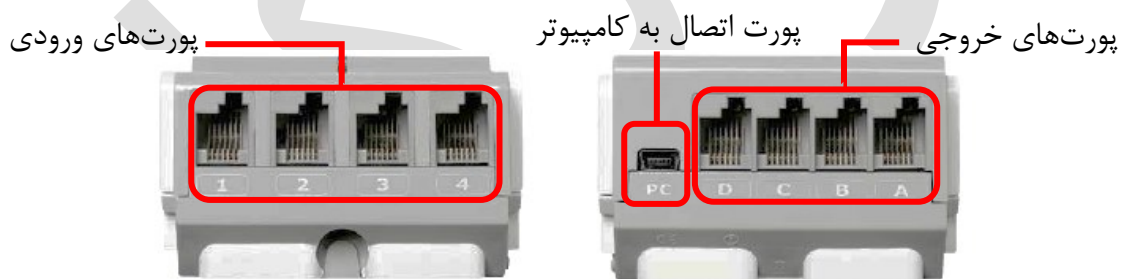
۲. منوی گردش در فایل‌ها (File Navigation): با استفاده از این منو می‌توانید به فایل‌های موجود در حافظه‌ی بریک دسترسی پیدا نمایید.

۳. منوی برنامه‌های بریک (Brick Apps): در این منو می‌توان به برنامه‌های گوناگونی که بر روی

بریک وجود دارد دسترسی پیدا کرد. در این قسمت به شرح مختصر هر یک از برنامه‌ها می‌پردازیم:

- نمایش پورت‌ها (Port View): برای مشاهده‌ی سنسور یا موتور متصل به هر پورت از این برنامه استفاده می‌توان نمود. در این صورت می‌توان با کلیدهای جهت، سنسور یا موتور متصل به هر پورت را مشاهده کرد. از این بخش می‌توان برای عیب‌یابی یا تنظیم سنسورها و موتورها استفاده کرد.

- کنترل موتور (Motor Control): برای کنترل موتورها به صورت دستی می‌توان از این برنامه استفاده نمود. در این صورت با فشردن کلید انتخاب می‌توان جفت موتور متصل به بریک را مشخص نمود و با کلیدها جهت، هر موتور را در دو جهت حرکت داد. برای بررسی موتورها و صحت از عملکرد آنها می‌توان از این برنامه استفاده نمود.
 - کنترل مادون قرمز (IR Control): از این برنامه برای تنظیم ریموت کنترل مادون قرمز می‌توان استفاده کرد. در این کتاب به شرح این کنترلر نمی‌پردازیم زیرا در بسته‌های اصلی ربات‌های EV3 این ریموت وجود ندارد.
 - برنامه‌نویسی بریک (Brick Program): از این برنامه می‌توان برای برنامه‌نویسی ربات استفاده نمود. از این برنامه می‌توان برای برنامه‌های ساده استفاده نمود اما به علت محیط ساده و محدودیت‌های آن، نمی‌توان برای برنامه‌های پیچیده استفاده کرد.
 - دیتالوگ بریک (Brick Datalog): از این برنامه برای ذخیره‌سازی و نمایش داده‌های ارسال شده از سنسورها می‌توان استفاده نمود.
۴. تنظیمات (Settings): در این منو می‌توان به تنظیمات بریک از جمله حجم صدا، تایمر خاموش شدن بریک، بلوتوث، وای‌فای و سایر اطلاعات بریک دسترسی داشت.



شکل ۲۰ پورت‌های ورودی و خروجی بریک

بر روی هر بریک ۸ پورت ورودی و خروجی قرار دارد. ۴ پورت در سمت بالا با نام‌های A، B، C و D قرار دارد که پورت‌های خروجی بوده و برای اتصال موتور به بریک می‌باشد. ۴ پورت دیگر در سمت پایین بلوک و با نام‌های ۱، ۲، ۳ و ۴ قرار دارد که پورت‌ها ورودی بوده و برای اتصال سنسورها به بریک می‌باشد. همچنین یک پورت Mini-USB برای اتصال بریک به کامپیوتر توسط کابل وجود دارد. برای اتصال بریک به کامپیوتر می‌توان از اتصال بی‌سیم نیز استفاده نمود.

موتورها



شکل ۲۱ موتورهای مجموعه‌ی EV3

در مجموعه‌ی EV3، دو نوع موتور وجود دارد که به نام‌های موتور بزرگ و موتور متوسط شناخته می‌شوند. این دو نوع موتور از نوع سروو می‌باشند که قادر به اندازه‌گیری میزان چرخش، سرعت و قدرت موتور را دارند. موتورهای سروو برای اندازه‌گیری میزان چرخش، از انکودر^۲ استفاده می‌نمایند. دقت اندازه‌گیری چرخش این موتور ۱ درجه می‌باشد. همچنین از سنسورهای موجود در این موتور برای استخراج اطلاعات در آزمایش‌ها می‌توان استفاده کرد.

فرق موتور بزرگ و موتور متوسط در قدرت، سرعت و نحوه‌ی قرارگیری آنهاست. موتور بزرگ برای قدرت‌های بیشتر اما سرعت‌های کمتر کاربرد دارد، زیرا حداکثر سرعت آن ۱۶۰-۱۷۰ دور بر دقیقه، گشتاور در حالت حرکت ۲۰ نیوتن بر سانتی‌متر و گشتاور در حالت سکون برابر ۴۰ نیوتن بر سانتی‌متر است. از طرفی برای موتور متوسط حداکثر سرعت آن برابر ۲۴۰-۲۵۰ دور بر دقیقه، گشتاور در حالت حرکت ۸ نیوتن بر سانتی‌متر و گشتاور در حالت سکون ۱۲ نیوتن بر سانتی‌متر است.

در واقع این موتورها عملگرهای ربات ما می‌باشند که می‌توان از آنها به عنوان محرک چرخ، گریپر، محرک تسمه و ... استفاده کرد. در ادامه به مقدمات کار با موتورها و چند نمونه از کاربردهای آن می‌پردازیم.

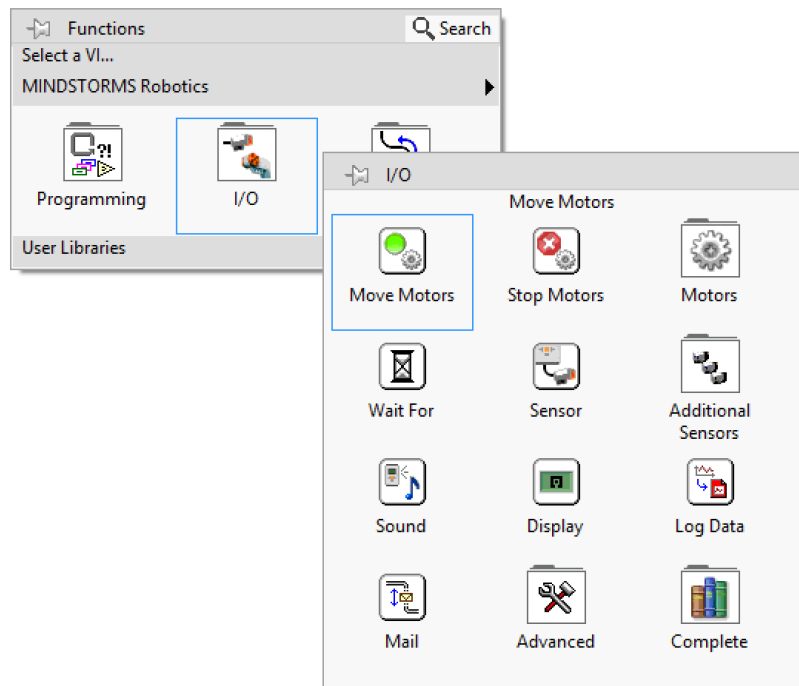
فرمان دادن به موتور

برای فرمان دادن به موتور در برنامه LabVIEW روش‌های مختلفی وجود دارد. ساده‌ترین نوع فرمان مشخص کردن قدرت/سرعت موتور می‌باشد. راه دیگر، مشخص کردن مقدار چرخش موتور است. راه‌های دیگری نیز برای حرکت موتور وجود دارد که بعداً به آنها می‌پردازیم.

فرمان با بلوک Move Motors

برای شروع یک موتور بزرگ را به پورت A بریک متصل نمایید. حال بریک را با کابل USB به کامپیوتر خود وصل نمایید. برای ساخت برنامه‌ای که به موتور فرمان بدهد، یک پروژه جدید تعریف نمایید. حال در پنجره بلوک دیاگرام، کلیک راست کرده و در بخش MINDSTORM Robotics به دسته‌ی I/O بروید و از آنجا بلوک Move Motors را انتخاب کنید و در فضای بلوک دیاگرام قرار دهید:

^۲ Encoder



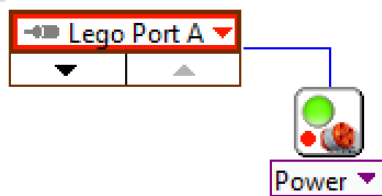
شکل ۲۲ انتخاب بلوک Move Motors در پنجره بلوک دیاگرام

بعد از قرار دادن بلوک Move Motors، بلوکی مانند زیر نمایان می‌گردد:



شکل ۲۳ بلوک Move Motors

با راست کلیک بر روی گرهی بالایی بلوک (Motors) و انتخاب Create Constant، پورت مورد نظر که موتور به آن متصل شده است را مشخص می‌کنید. در اینجا می‌خواهیم این فرمان به موتور موجود در پورت A اعمال شود، پس بعد از انتخاب Create Constant از منوی کشویی ایجاد شده، Lego Port A را انتخاب می‌کنیم:

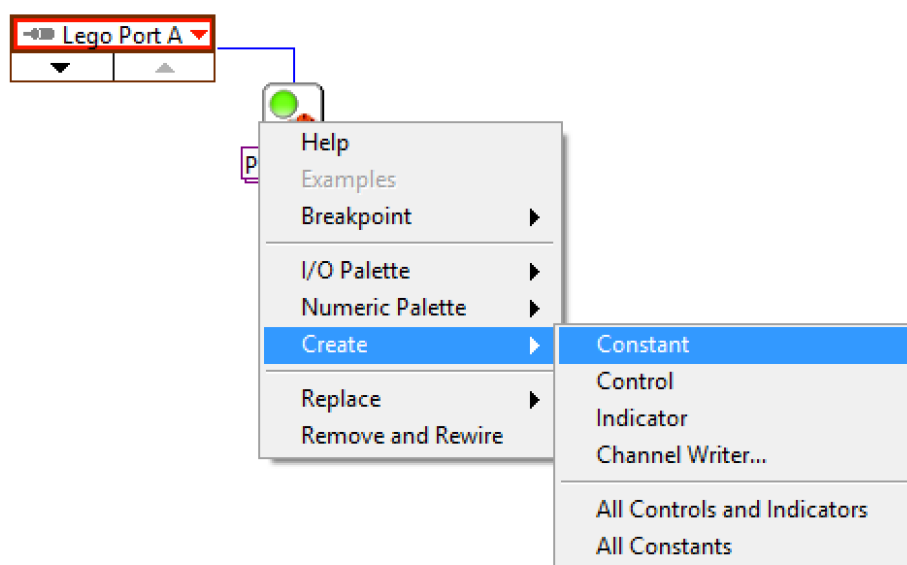


شکل ۲۴ اتصال موتور مورد نظر به گرهی Motors

دقت کنید که اگر موتوری را در این قسمت مشخص ننمایید، برنامه این فرمان را به تمام موتورهایی که به بریک متصل است اعمال می‌کند.

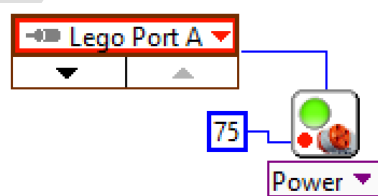
در قدم بعد نوبت به مشخص کردن مقدار قدرت/سرعت موتور می‌باشد. همانطور که در بلوک مشخص است شما می‌توانید از منوی کشویی موجود در پایین بلوک، نوع فرمان را مشخص کنید. این بلوک دارای دو نوع فرمان به نام‌های Constant Power یا قدرت ثابت و Constant Speed یا سرعت ثابت می‌باشد. همانطور که از نام‌های آنها مشخص است در گزینه‌ی اول، ما قدرت موتور را مشخص می‌کنیم که با توجه به مقدار باری که بر روی موتور وجود دارد سرعت چرخش آن تغییر خواهد کرد. اما در گزینه‌ی دوم سرعت موتور مشخص می‌شود، در این حالت موتور با سنسوری که دارد سرعت خود را مطابق با سرعت فرمان داده شده تنظیم می‌کند.

برای مشخص کردن قدرت/سرعت موتور در این بلوک، باید یک بلوک عدد ثابت را به گره‌ی ورودی آن (در برنامه با نام Power/Speed 1 شناخته می‌شود) متصل نماییم که مبین قدرت/سرعت چرخش موتور موردنظر می‌باشد. برای سادگی می‌توان بر روی گره‌ی ورودی آن راست کلیک کرده و گزینه‌ی Create → Constant را انتخاب کنیم.



شکل ۲۵ ایجاد عدد ثابت بر روی گره‌ی Power 1

با انتخاب بلوک عدد ثابت مانند تصویر زیر، به طور پیش فرض عدد ثابت ۷۵ ایجاد می‌شود:

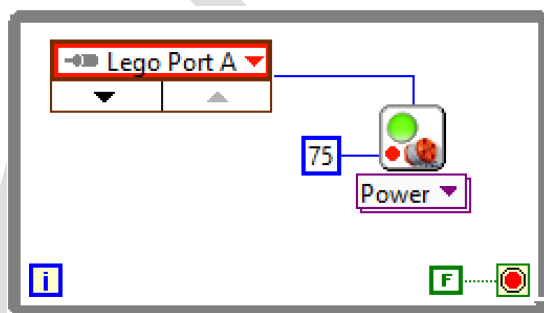


شکل ۲۶ اختصاص دادن مقدار ۷۵ به عنوان قدرت موتور متصل به پورت A

عدد داده شده به عنوان قدرت/سرعت به این بلوک باید بین 100- و 100 باشد. عددهای منفی به معنی چرخش در جهت عکس می‌باشد. اگر عددهایی بیشتر از 100 و یا کمتر از 100- به بلوک داده شوند به عنوان همان 100 و 100- تلقی می‌شوند. در واقع این عدد مشخص کننده درصد قدرت/سرعت حرکت موتور می‌باشد

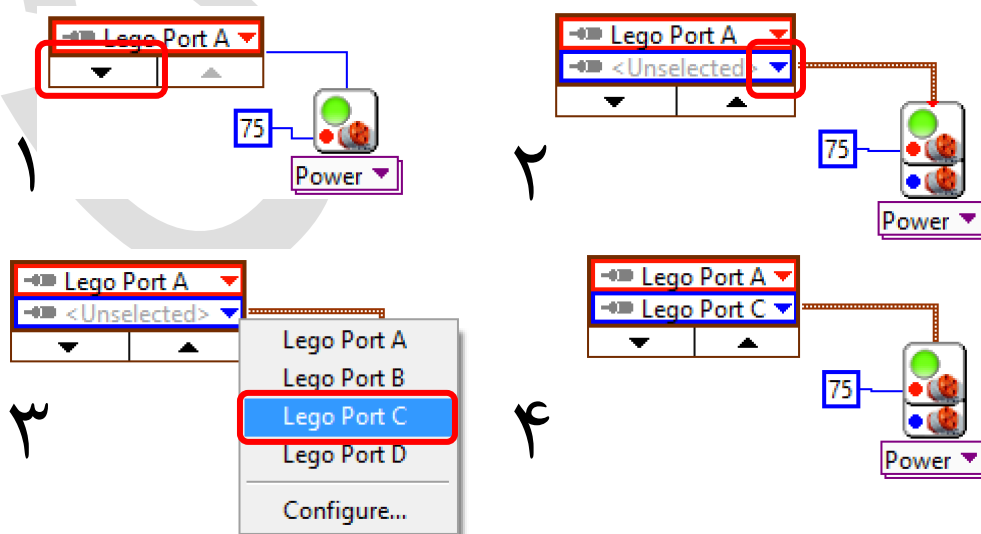
که به طور پیش فرض بر روی ۷۵٪ یا $\frac{3}{4}$ است. حال با آپلود و اجرای برنامه، موتور متصل به پورت A با قدرت ۷۵، به حرکت درمی‌آید.

با اجرای این برنامه، موتورها در یک لحظه فعال شده اما به سرعت متوقف می‌شوند و برنامه به پایان می‌رسد، به این دلیل که ما تنها در لحظه‌ی اولیه، فرمان حرکت را صادر کرده‌ایم و بعد از اجرای این فرمان، برنامه به پایان رسیده و موتورها متوقف خواهند شد. برای حرکت دائمی موتورها باید این فرمان به طور مستمر به ربات ارسال شود در نتیجه لازم است تا به طور مداوم این فرمان را اجرا کنیم. برای اینکار، می‌توان از یک حلقه‌ی بینهایت While استفاده کرد. همانطور که در فصل‌های قبل توضیح داده شد با ایجاد حلقه‌ی بینهایت While می‌توان یک یا چند بلوک را به طور بینهایت و برای همیشه اجرا کرد. پس بعد از اعمال حلقه‌ی بینهایت خواهیم داشت:



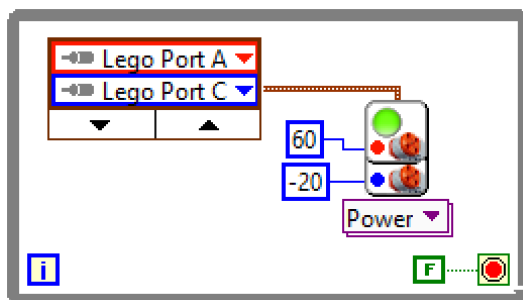
شکل ۲۷ ایجاد حلقه‌ی بینهایت برای تکرار مداوم حرکت موتور

حال اگر بخواهیم به طور همزمان به چند موتور فرمان بدهیم می‌توان از یک بلوک Move Motors برای فرمان دادن به چند موتور استفاده نماییم. به این صورت که بر روی فلش ظاهر شده در زیر Lego Port A کلیک کرده و از منوی کشویی جدید ایجاد شده، پورت سایر موتورها را انتخاب می‌نماییم. برای مثال اگر یک موتور بر روی پورت A و موتور دیگر بر روی پورت C باشد داریم:



شکل ۲۸ نحوه اضافه کردن موتور به گره‌ی Motors

در این حالت قادر خواهیم بود تا قدرت/سرعت دو موتور را به صورت مجزا نیز تعیین نماییم. برای اینکار مشابه قبل یک بلوک عدد ثابت به گره‌ی Power/Speed 2 متصل می‌کنیم:



شکل ۲۹ حرکت دو موتور در حلقه بینهایت

با آپلود و اجرای این برنامه، موتور متصل به پورت A به قدرت ۶۰٪ و موتور متصل به پورت C با قدرت ۲۰٪ در جهت عکس به حرکت درمی‌آیند.

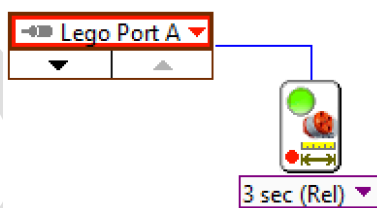
فرمان با بلوک Fixed Distance

این بلوک برای ایجاد حرکت با فاصله‌ی ثابت استفاده می‌شود. کاربرد این بلوک در مواردی است که از قبل مقدار حرکت ربات یا مقدار چرخش موتور را می‌دانیم و یا اینکه می‌توانیم محاسبه کنیم. برای اضافه کردن این بلوک به برنامه‌ی خود، به بخش MINDSTORM Robotics رفته و از قسمت I/O → Motors → Fixed Distance را انتخاب می‌نماییم:



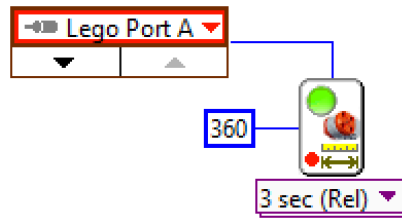
شکل ۳۰ بلوک Fixed Distance

در ابتدا در این بلوک مانند بلوک Move Motors پورت(های) متصل به موتور(های) مورد نظر را مشخص می‌کنیم:



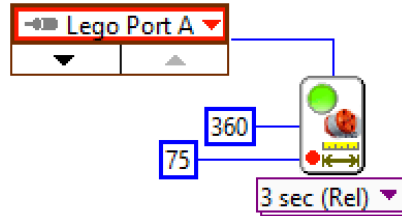
شکل ۳۱ اتصال گره‌ی Motors به پورت مورد نظر

در قدم بعد باید مقدار دوران و سرعت دوران موتور را مشخص کنیم. برای موتورهایی که در مجموعه EV3 قرار دارند، ما قادر خواهیم بود تا با دقت ۱ درجه آنها را جابجا نماییم. پس عددی که به گره 1 Distance می‌رود مبین مقدار چرخش موتور به درجه است. مقدار چرخش به طور پیش فرض ۳۶۰ درجه در نظر گرفته شده است. با ایجاد یک بلوک عدد ثابت بر روی این گره می‌توانیم مقدار چرخش موتور را مشخص کنیم:



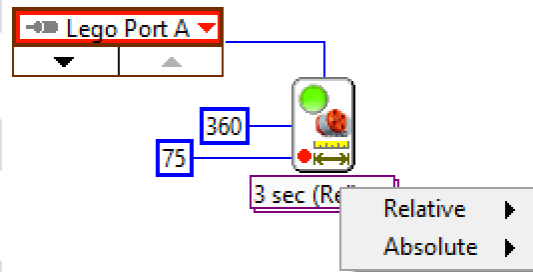
شکل ۳۲ تخصیص دادن مقدار زاویه‌ی چرخش موتور

گره‌ی Speed 1 برای تعیین سرعت دوران موتور می‌باشد که همانند بلوک Move Motors تعیین می‌گردد. با اعمال یک عدد ثابت بین ۱۰۰- و ۱۰۰+ به گره‌ی Speed 1، می‌توان سرعت چرخش موتور را تعریف کرد:



شکل ۳۳ تخصیص دادن مقدار قدرت موتور

تفاوت دیگری که این بلوک با بلوک Move Motors دارد در منوی کشویی پایین بلوک است. در این بلوک با باز کردن منوی کشویی، دو گزینه نسبی (Relative) و مطلق (Absolute) نمایان می‌گردد.



شکل ۳۴ مشخص کردن حرکت نسبی یا مطلق موتور در منوی کشویی پایین بلوک

در حرکت نسبی، موتور به اندازه‌ی مشخص شده از موقعیت کنونی خود دوران می‌کند اما در حرکت مطلق، موتور با توجه به مقدار خوانده شده توسط انکودر خود، موتور را به موقعیت گفته شده جابجا می‌کند. برای مثال اگر در گره‌ی Distance 1 مقدار ۲۰۰ را مشخص کنیم و مقدار خوانده شده‌ی انکودر موتور در حالت اولیه ۱۰۰ باشد، در حالت حرکت نسبی، موتور در نهایت به موقعیت ۳۰۰ می‌رسد اما در حالت حرکت مطلق، موتور به همان موقعیت ۲۰۰ جابجا می‌گردد. در واقع می‌توان گفت در حالت نسبی و مطلق، تعریف بلوک از گره‌ی Distance 1 تغییر می‌کند.

لازم به ذکر است که برای حرکت نسبی موتور، باید مقدار جابجایی و سرعت، هر دو مثبت یا منفی باشند تا بلوک به درستی عمل کند.

در ادامه پس از تعیین نوع حرکت که نسبی یا مطلق است، باید مقدار توقف برنامه را برای اجرای این دستور مشخص نماییم. به طور پیش فرض این بلوک به صورت نسبی موتور را جابجا کرده و برای اجرای این فرمان ۳ ثانیه مهلت می‌دهد. اگر سرعت موتور کم باشد یا مقدار دوران زیاد باشد، بیش از ۳ ثانیه برای دوران به میزان

تعیین شده لازم است، در نتیجه باید این زمان را افزایش داد. در صورتی که مقدار دوران مشخص شده زودتر از زمان تعیین شده انجام شود، برنامه به طور خودکار جریان پیدا کرده و ادامه می‌یابد.

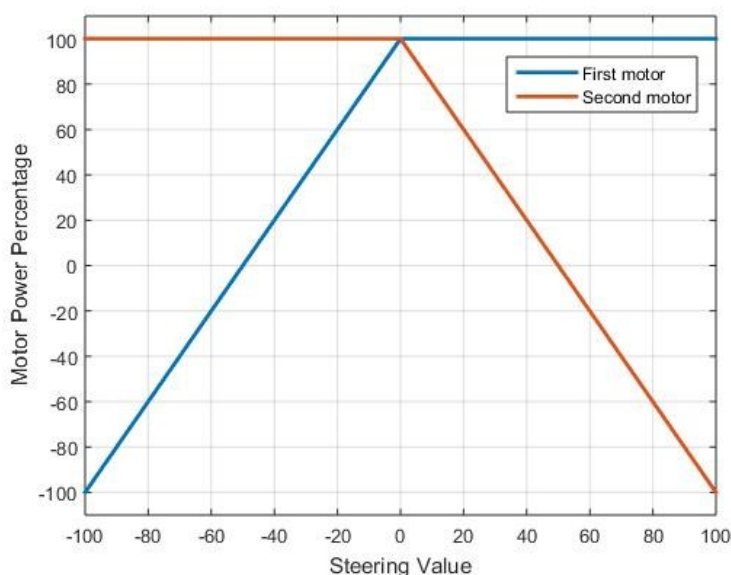
بلوک LEGO Steering

یکی از مهمترین فرمان‌هایی که می‌توان به موتورها داد برای پیمودن مسیرهای منحنی و دایروی است. برای اینکار می‌توان از بلوک‌های Move Motors و یا Fixed Distance استفاده کرد، اما برای سادگی و سرعت بیشتر در برنامه‌ریزی ربات، بلوک LEGO Steering طراحی شده است. این بلوک در قسمت Functions → I/O → Motors → MINDSTORM Robotics با نام LEGO Steering قابل دسترسی است:



شکل ۳۵ بلوک LEGO Steering

در این بلوک با مشخص کردن دو موتور (دو چرخ محرک ربات) در گرهی Motors، می‌توان عملیات فرمان دادن را با قدرت‌های مختلف موتور انجام داد. به اینصورت که در گرهی Power 1 مقدار قدرت موتور را که عددی بین ۱۰۰- و ۱۰۰+ است انتخاب کرده و در گرهی Steering 1 عددی بین ۱۰۰- و ۱۰۰+ اختصاص داده که در این حالت قدرت موتور اول و دوم (به ترتیبی که در گرهی Motors مشخص شده‌اند) درصدی از قدرت مشخص شده در گرهی Power 1 می‌باشد. این درصد طبق نمودار زیر مشخص می‌شود:



شکل ۳۶ نمودار قدرت هر موتور نسبت به مقدار ورودی گرهی Steering

در واقع در مقادیر مثبت برای گرهی Steering 1، موتور اول قدرتی برابر با مقدار داده شده در گرهی Power 1 داشته و موتور دوم درصدی از این قدرت را اختیار می‌کند. برای مقادیر منفی این عمل برای دو موتور عکس می‌باشد. در بخش بعدی به طور دقیق به کاربرد این بلوک در برنامه‌ریزی ربات خواهیم پرداخت.

بلوک Stop Motors

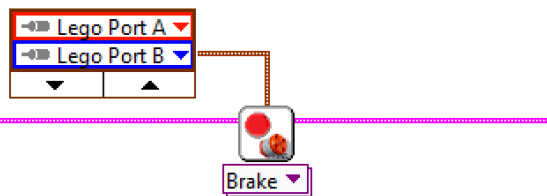
دو بلوکی که در قسمت‌های قبل معرفی شدند برای فرمان دادن و به حرکت درآوردن موتورها استفاده می‌شود که با توجه به نوع حرکت و هدف ما از آن حرکت، می‌توان از یکی از آنها استفاده کرد. اما نکته‌ای که باید توجه کرد این است که هنگامی که به موتورها فرمان حرکت می‌دهیم، بعد از اتمام حرکت باید به موتورها فرمان توقف را نیز صادر کنیم. زیرا هنگامی که از چند فرمان حرکت موتور استفاده می‌کنیم، در صورتیکه از بلوک Stop Motors استفاده نشود، فرمان‌ها با یکدیگر تداخل داشته و معمولاً حرکت از پیش تعیین شده توسط ربات اجرا نخواهد شد.

بلوک Stop Motors از آدرس `Stop Motors → I/O → MINDSTORM Robotics → Functions` قابل دسترسی است:



شکل ۳۷ بلوک Stop Motors

برای استفاده از این بلوک تنها کافیست پورت(های) متصل به موتور(ها) را مشخص نماییم و سپس گره‌های ورودی و خروجی NXT/EV3 را به ترتیب به بلوک‌های قبل و بعد متصل کنیم:



شکل ۳۸ اتصال گره‌های بلوک Stop Motors

توجه کنید که اگر گره‌ی Motors را مشخص نکنیم، این بلوک فرمان توقف را به تمامی موتورهای متصل به بریک صادر می‌کند.

گزینه‌ی دیگری که برای این بلوک وجود دارد از منوی کشویی پایین بلوک قابل دسترسی است. این بلوک دارای دو حالت Brake و Coast است. در حالت Brake موتورها به سرعت متوقف می‌شوند اما در حالت Coast سرعت موتورها به آهستگی کم شده و سپس متوقف می‌شوند که بسته به هدف ما می‌تواند این گزینه انتخاب شود.

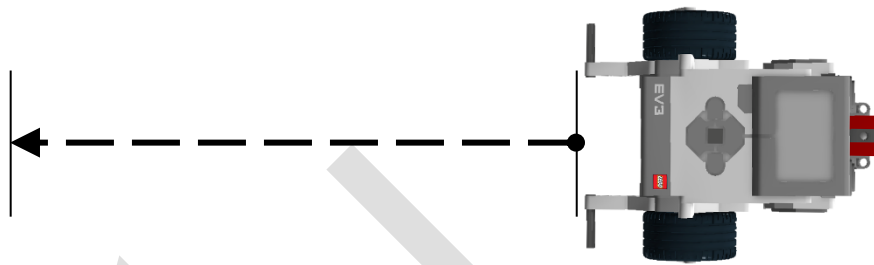
کاربرد موتورها

همانطور که گفته شد از موتورها برای بخش‌های مختلف یک ربات از جمله گریپر، چرخ، بازو، پرتاب کننده، محرک تسمه و ... می‌توان استفاده نمود. در واقع عملگرهای ما در مجموعه EV3، این موتورهای DC می‌باشند اما در صنعت از عملگرهای مختلفی به جز موتور DC مانند سیلندرهای هیدرولیکی و نیوماتیکی نیز استفاده می‌شود که هر کدام از این عملگرها ویژگی خاص خود را دارند.

یکی از ساده‌ترین و پرکاربردترین استفاده موتور در ربات‌ها، به عنوان محرک چرخ می‌باشد. در این حالت با چرخش موتور، چرخ نیز دوران کرده و باعث حرکت ربات بر روی زمین می‌گردد.

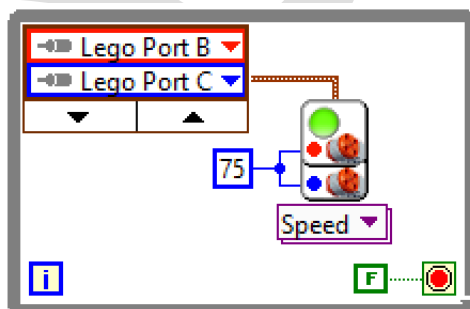
ربات پایه Educator را طبق توضیحات داده شده در پیوست کتاب بسازید. در این ربات دو چرخ محرک داریم که به دو موتور متصل شده‌اند. یکی از موتورها به پورت B و دیگری به پورت C بریک وصل شده‌اند. با فعال کردن این دو موتور می‌توان مسیره‌های مختلفی را توسط ربات پیمود. در ادامه برنامه‌ریزی برای چند مسیر اصلی را بررسی می‌کنیم.

(۱) حرکت در مسیر مستقیم



شکل ۳۹ حرکت ربات در مسیر مستقیم

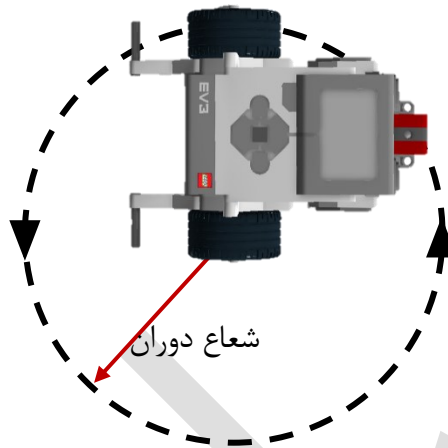
همانطور که می‌دانیم برای حرکت در مسیر مستقیم لازم است تا هر دو چرخ با یک سرعت برابر دوران کنند، در این حالت تمامی نقاط ربات دارای سرعت یکسان و برابر می‌باشند. برای این منظور می‌توان از بلوک Move Motors استفاده نمود. مانند آنچه که در قسمت قبل توضیح داده شد بلوک Move Motors را به سیستم اضافه کرده و فرمان را به دو موتور B و C اعمال می‌کنیم:



شکل ۴۰ حرکت مداوم در مسیر مستقیم با بلوک Move Motors

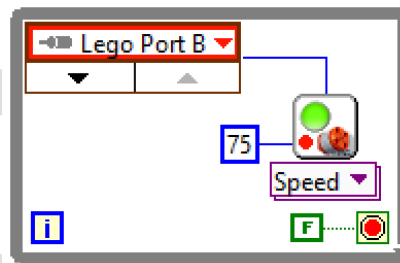
دقت کنید که در بلوک Move Motors از منوی کشویی Constant Speed یا سرعت ثابت را انتخاب کرده و سرعت هر دو موتور (گره Speed 1 و Speed 2) را مساوی و برابر ۷۵ قرار داده‌ایم. همچنین برای اجرای مداوم این فرمان، آنرا درون یک حلقه‌ی بینهایت While قرار داده‌ایم. با تغییر عدد ثابت ۷۵ بین ۱۰۰- و ۱۰۰ می‌توان سرعت حرکت بر روی مسیر مستقیم را کم و زیاد نمود. سعی کنید این برنامه را با بلوک LEGO Steering نیز انجام دهید. بلوک LEGO Steering معمولا برای چرخش ربات و پیمودن مسیره‌های منحنی استفاده می‌گردد اما می‌توان حرکت مستقیم را نیز فرمان داد.

۲) چرخش حول یک چرخ



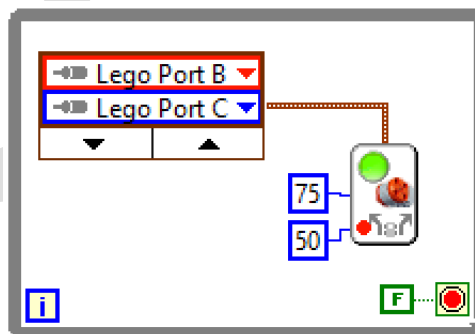
شکل ۴۱ چرخش ربات حول یک چرخ

برای چرخش حول یک چرخ، تنها کافیست تا فقط یک چرخ را به حرکت درآوریم. برای مثال اگر بخواهیم حول چرخ C چرخش کنیم، لازم است تا فرمان Move Motors را تنها به موتور B اعمال کنیم.



شکل ۴۲ چرخش مداوم حول یک چرخ با بلوک Move Motors

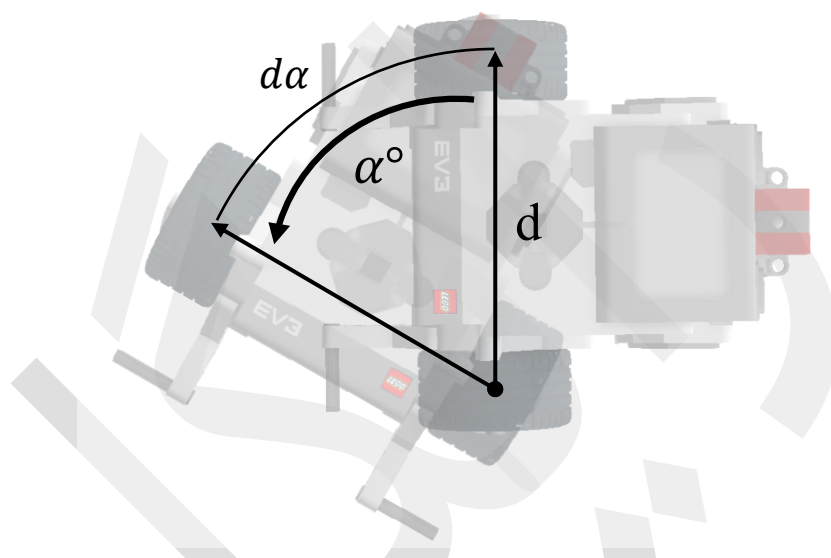
اگر بخواهیم چرخش حول یک چرخ را با بلوک LEGO Steering انجام دهیم با توجه به نمودار این بلوک، می‌توان با قرار دادن Power 1 برابر ۷۵ و Steering 1 برابر ۵۰، دقیقاً همین عمل را برنامه‌ریزی نماییم:



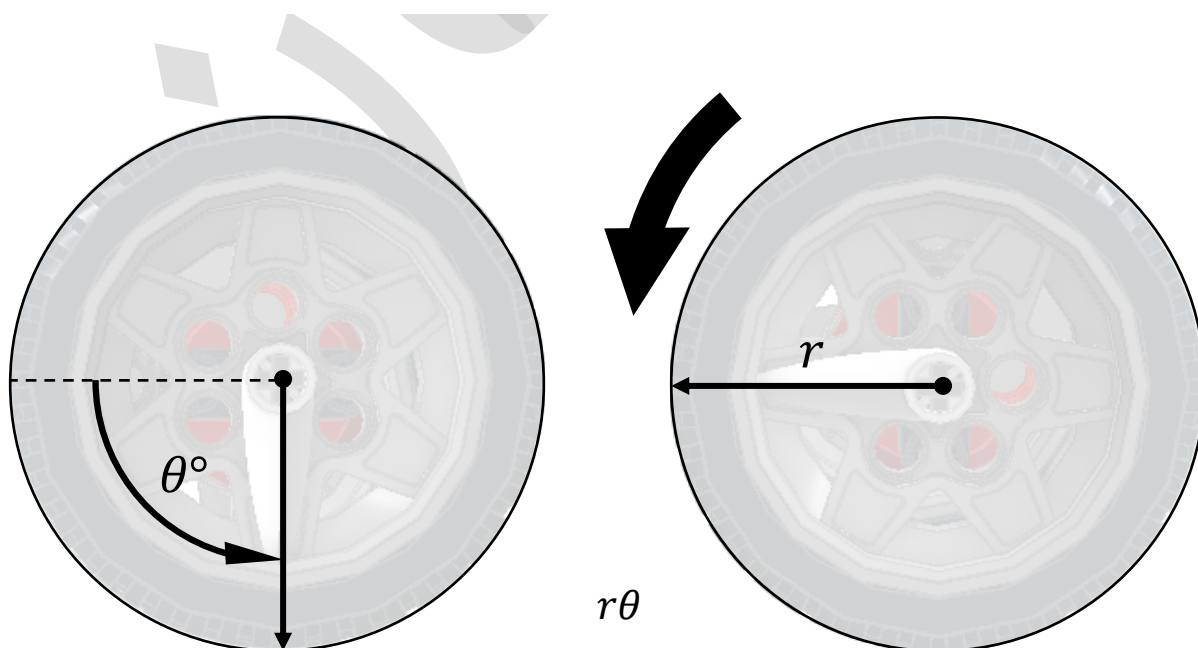
شکل ۴۳ چرخش مداوم حول یک چرخ با بلوک LEGO Steering

با اجرای برنامه‌ی فوق، ربات شروع به چرخش حول چرخ C کرده و تا بینهایت این حرکت را تکرار می‌کند. اما در بسیاری از کاربردها، لازم است تا ربات ما به اندازه‌ی مشخصی دوران کرده و سپس عملیات دیگری را دنبال کند، در نتیجه لازم است تا چرخش حول یک چرخ را به مقدار مشخصی انجام دهیم. برای اینکار از بلوک Fixed Distance استفاده می‌کنیم تا با چرخش یک چرخ به مقدار مشخص، تغییر زاویه ربات را تعیین

کنیم. همانطور که از آزمایش قبل متوجه شدید، چرخش حول یک چرخ مانند حرکت ربات بر روی دایره‌ای به شعاع فاصله دو چرخ و مرکز چرخ ثابت است، پس برای تعیین زاویه‌ی دوران، لازم است تا طول مسیری که بر روی این دایره می‌پیماییم را محاسبه کنیم. از طرفی ما با فرمان Fixed Distance، زاویه چرخش موتور و چرخ را کنترل می‌نماییم. اگر شعاع چرخ (که می‌توان با اندازه‌گیری بدست آورد) را برابر r در نظر گرفته و زاویه چرخش موتور را θ° بگیریم، مسیر پیموده شده توسط این چرخ برابر است با $r\theta$. از طرفی اگر زاویه چرخش ربات را α فرض کنیم و فاصله‌ی دو چرخ نیز d باشد، مقدار $d\alpha$ برابر با طول مسیر دایروی پیموده شده است:



شکل ۴۴ مسیر طی شده با حرکت حول یک چرخ

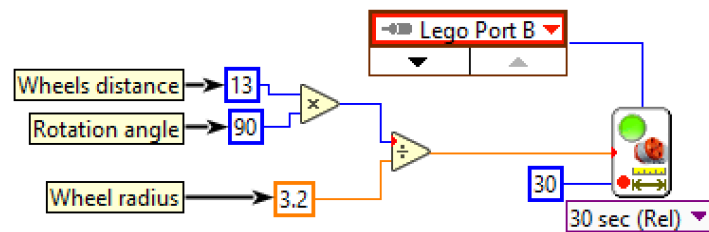


شکل ۴۵ مسیر طی شده توسط یک چرخ

با مساوی قرار دادن این دو مسیر پیموده شده (مسیر پیموده شده توسط چرخ و مسیر پیموده شده بر روی دایره)، می‌توان مقدار θ را بدست آورد:

$$r\theta = d\alpha \rightarrow \theta = \frac{d\alpha}{r}$$

در عمل برای استفاده از این فرمول می‌توان مقادیر r و d را توسط اندازه‌گیری با متر، خط‌کش یا کولیس بدست آورد و مقدار زاویه چرخش ربات (α) را نیز طبق خواسته خود می‌دانیم. برای ساده تر شدن محاسبات می‌توان رابطه‌ی گفته شده را در برنامه نوشت تا تنها با تغییر زاویه‌ی چرخش، زاویه دوران چرخ محاسبه گردد:



شکل ۴۶ چرخش ۹۰ درجه حول یک چرخ با بلوک Fixed Distance

در اینجا مقدار چرخش مورد نظر ما ۹۰ درجه است. برای تغییر این زاویه کفایت تا مقدار Rotation angle را برابر با زاویه مطلوب قرار دهیم. دقت کنید که از منوی کشویی پایین بلوک Fixed Distance گزینه‌ی Relative \rightarrow Wait Up To \rightarrow 30 sec خود جابجایی θ را داشته باشد ثانیاً به موتور حداکثر زمان ۳۰ ثانیه را بدهیم تا زاویه خود را تغییر دهد. در صورتیکه چرخش در کمتر از ۳۰ ثانیه انجام شود، برنامه ادامه یافته و پایان می‌یابد.

این برنامه شامل دو بلوک که در دو قسمت قبل توضیح داده شده است می‌باشد که با گره‌های خروجی و ورودی NXT/EV3 بلوک‌های مجاور به یکدیگر متصل شده‌اند. همانطور که گفته شد برای حرکت صحیح موتورها و عدم تداخل فرمان‌های موتور با یکدیگر، از بلوک Stop Motors بعد از هر فرمان موتور استفاده شده است.

دانشگاه

سنسور تماس



شکل ۴۹ سنسور ضربه مجموعه‌ی EV3

سنسور تماس (یا لمس Touch) یک دکمه‌ی ساده است که برای شناسایی برخورد ربات با اجسام و موانع و یا به عنوان یک ورودی صفر و یک طراحی شده است. این سنسور قابلیت این را دارد که فشرده شدن، رها شدن، ضربه خوردن دکمه را تشخیص دهد و همچنین تعداد دفعات دکمه زدن تکی و چندتایی را بشمارد. از مهمترین کاربرد این سنسور ساخت ربات‌های حل هزارتو می‌باشد.

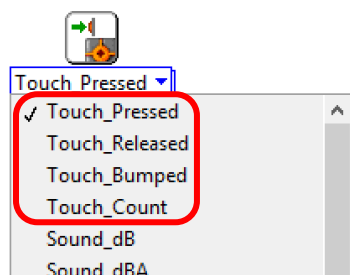
خواندن سنسور تماس

ابتدا بلوک Sensor از بخش I/O → MINDSTORMS Robotics → functions را به بلوک دیاگرام خود اضافه می‌کنیم.



شکل ۵۰ بلوک Sensor

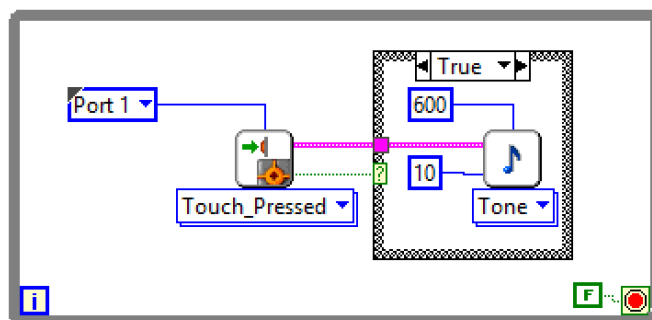
برای این سنسور ۴ مد یا حالت کاری تعریف شده است. خروجی بلوک Sensor در هر ۴ مد این سنسور، به صورت یک متغیر منطقی True/False خواهد بود.



شکل ۵۱ چهار مد تعریف شده برای سنسور تماس

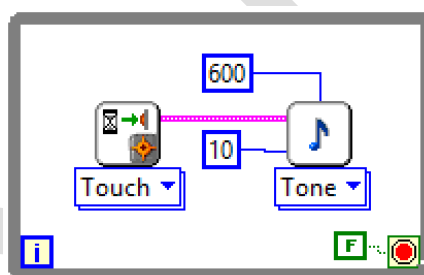
این ۴ مد به ترتیب Pressed, Released, Bumped, و Count می‌باشند.

- در مد Pressed، هرگاه که دکمه فشرده شود، گرهی خروجی True و در غیر اینصورت False را تولید می‌کند. با قرار دادن این بلوک در حلقه‌ی بینهایت و همچنین اضافه کردن بلوک Sound می‌توان برنامه‌ی زیر را نوشت که با فشردن دکمه، بریک بوقی را ایجاد کند.



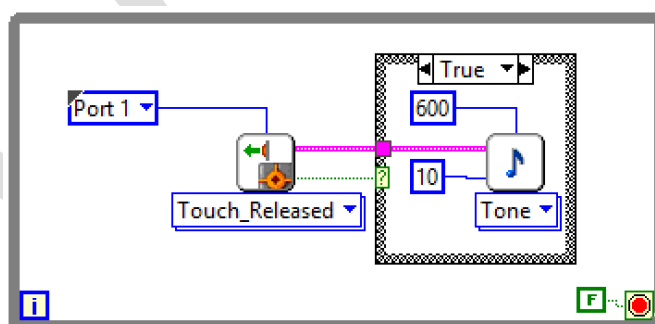
شکل ۵۲ ایجاد صدای بوق با فشردن دکمه‌ی سنسور

همچنین می‌توان چنین برنامه‌ای را با بلوک Wait For نیز نوشت:



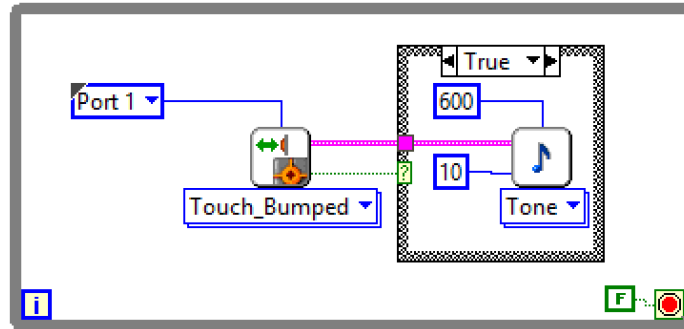
شکل ۵۳ ایجاد صدای بوق در هنگام فشردن دکمه‌ی سنسور با بلوک Wait For

- مد Released مشابه مد Pressed می‌باشد با این تفاوت که عکس آن عمل می‌کند. یعنی در حالت Released، با فشرده شدن دکمه، گرهی خروجی False و در سایر حالات True می‌باشد. اگر برنامه‌ای مشابه برنامه‌ای که برای مد قبل نوشتیم بنویسیم آنگاه بریک همواره در حال بوق زدن می‌باشد مگر آنکه دکمه‌ی آن فشرده شود:



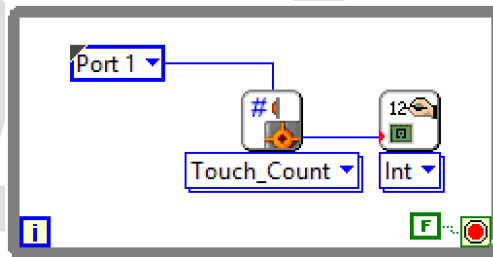
شکل ۵۴ قطع صدای بوق با فشردن دکمه‌ی سنسور

- در مد Bumped، با هربار ضربه خوردن به دکمه (در لحظه‌ی رها شدن)، گرهی خروجی یک پالس True ایجاد می‌کند. فرق این حالت با دو حالت قبل این است که در حالات قبل گرهی خروجی به طور پیوسته True می‌ماند اما در این حالت تنها یک لحظه True می‌شود و در سایر مواقع همواره False است. در برنامه‌ی زیر، با هربار ضربه خوردن به دکمه، بریک بوقی را تولید می‌کند.



شکل ۵۵ ایجاد صدای بوق با ضربه خوردن به دکمه‌ی سنسور

- مد آخر این سنسور Count می‌باشد. در این مد، خروجی گره یک عدد می‌باشد که با هر بار فشردن دکمه، سنسور اقدام به شمارش می‌کند. در واقع در این حالت تعداد دفعات دکمه زدن شمارش می‌شود. در برنامه‌ی زیر با هر بار فشردن دکمه، تعداد دفعات شمارش شده افزایش یافته و بر روی نمایشگر بزرگ نشان داده می‌شود:



شکل ۵۶ نمایش تعداد دفعات فشرده شدن دکمه‌ی سنسور بر روی نمایشگر بزرگ

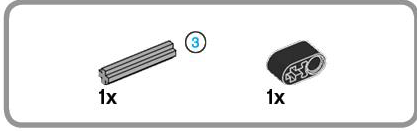
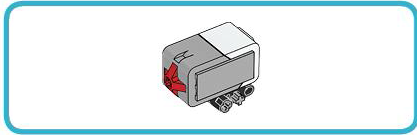
در یک جمع بندی کلی و برای شفاف شدن تفاوت بین هر سه مد (Pressed, Released و Bumped)، در جدول زیر، خروجی بلوک در شرایط مختلف آورده شده است:

			خروجی وضعیت
Bumped	Released	Pressed	
False	True	False	دکمه رها شده
False	False	True	دکمه فشرده شده
True	True	False	لحظه رها شدن دکمه

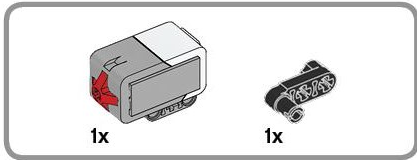
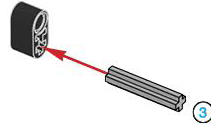
جدول ۱ وضعیت گره‌ی خروجی در سه مد سنسور تماس

کاربردهای سنسور تماس

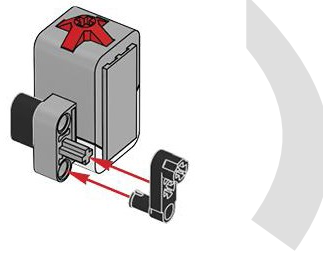
یکی از ساده‌ترین و پرکاربردترین استفاده‌ی این سنسور، برای تعیین برخورد ربات با موانع روبروی خود می‌باشد. برای این منظور ابتدا به صورت زیر، سنسور تماس را بر روی ربات Educator متصل می‌کنیم:



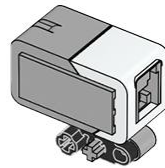
1



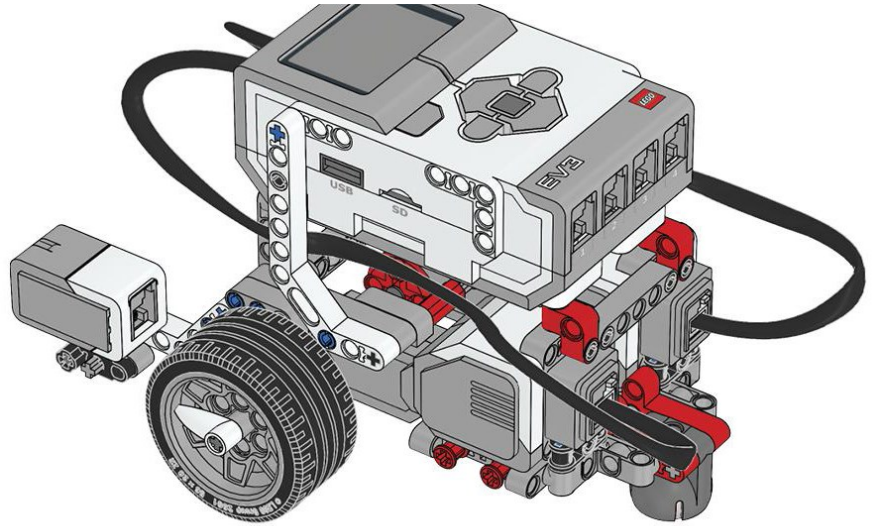
2



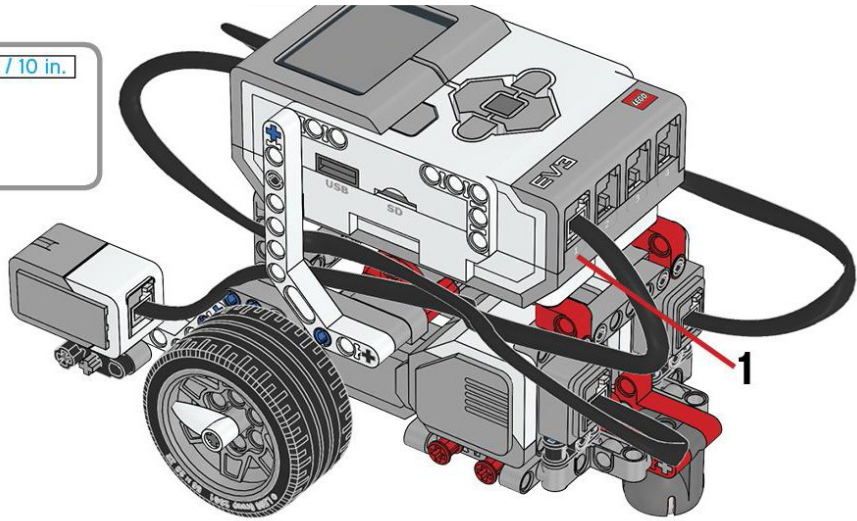
3



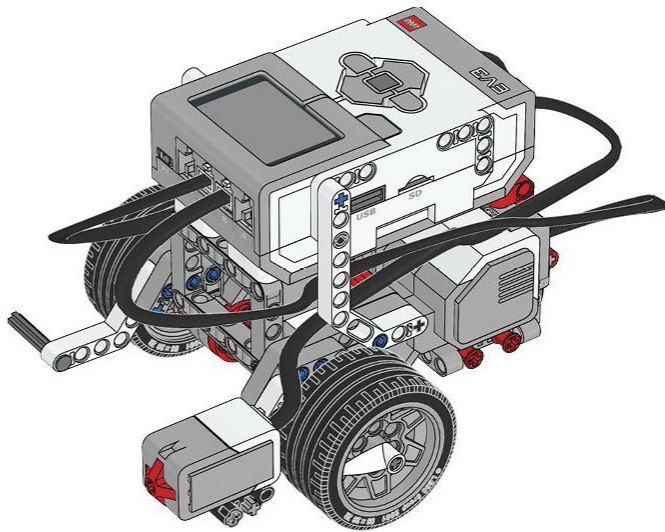
4

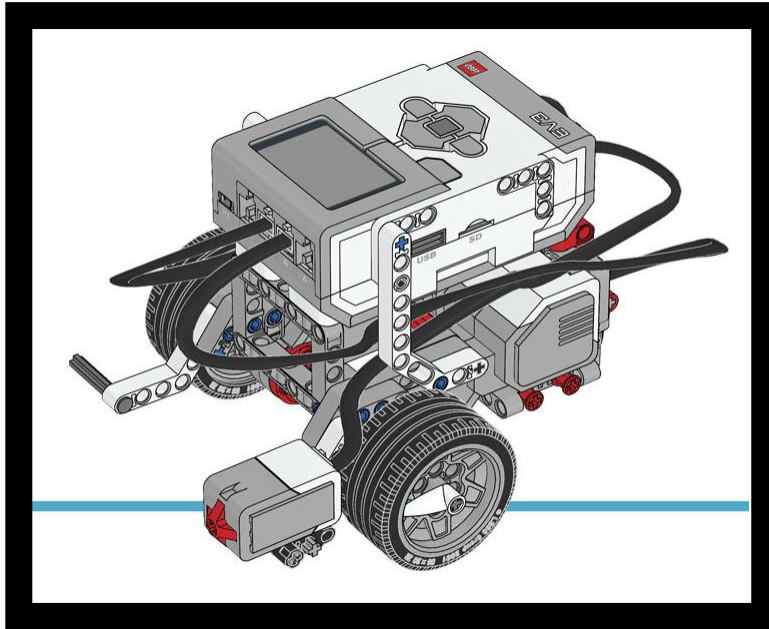


5



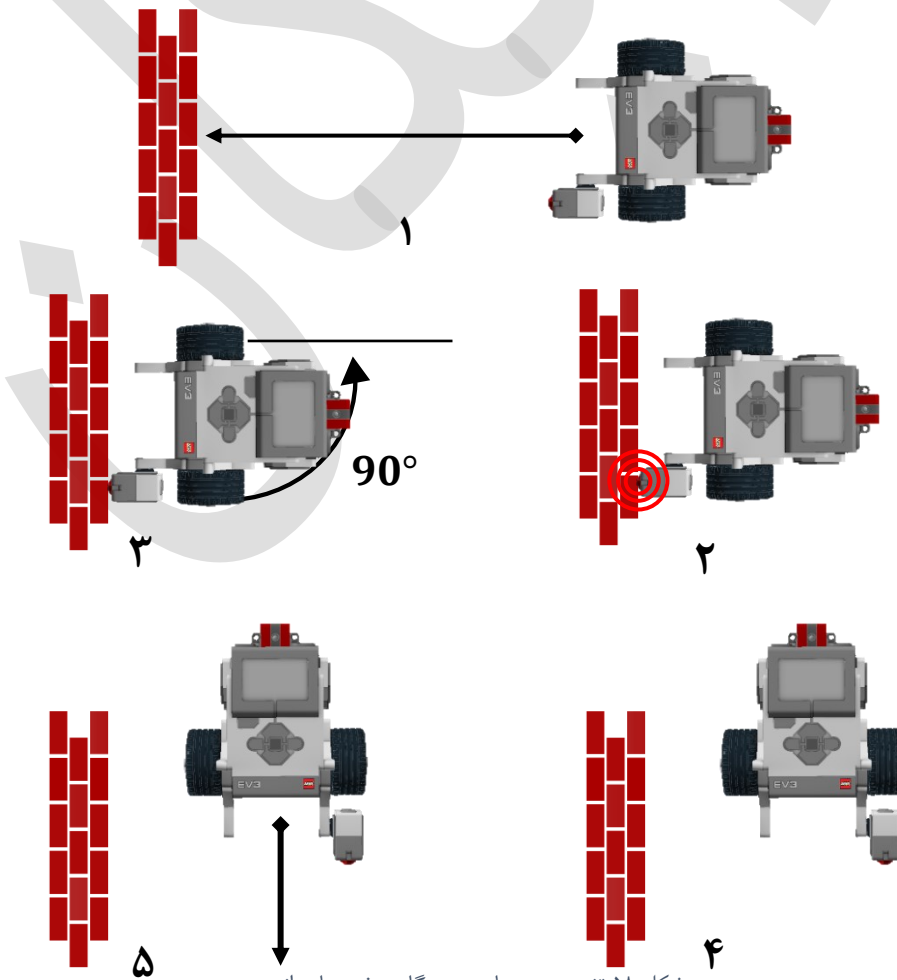
6





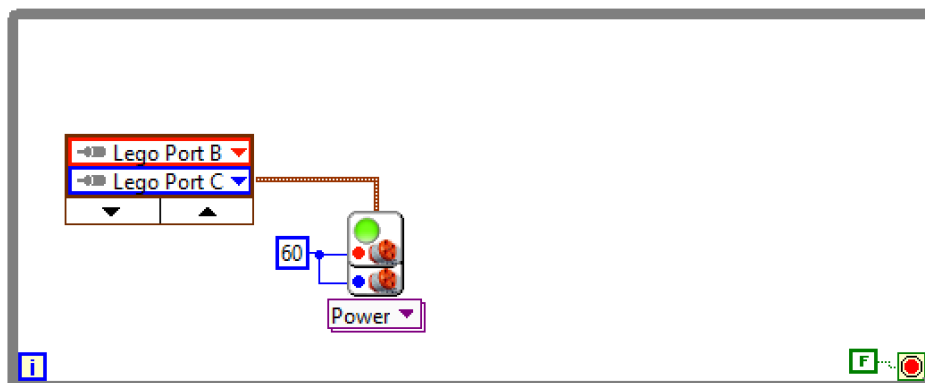
شکل ۵۷ نصب سنسور تماس بر روی ربات Educator

هدف ما برنامه‌نویسی رباتی است که مسیر مستقیم را طی کند و در صورت برخورد به مانع، چرخش انجام دهد و سپس دوباره به راه خود ادامه دهد.



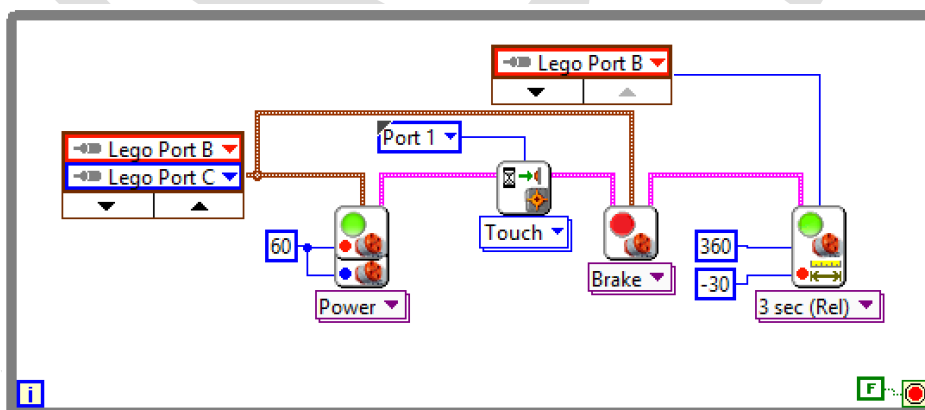
شکل ۵۸ تغییر مسیر ربات در هنگام برخورد با موانع

برای این منظور ابتدا بلوک Move Motor را درون حلقه‌ی بینهایت قرار می‌دهیم تا ربات به طور پیوسته به حرکت مستقیم خود ادامه دهد:



شکل ۵۹ ایجاد حرکت مداوم در مسیر مستقیم

در ادامه می‌خواهیم که اگر سنسور تماس فشرده شد ربات متوقف شده و بر روی یک چرخ خود به صورت عقب‌گرد بچرخد (اگر درجا یا رو به جلو بچرخد مسلماً مانع روبرو، اجازه‌ی چرخش را نمی‌دهد). برای تشخیص فشرده شدن دکمه از بلوک Wait For استفاده می‌کنیم و آنرا بر روی Wait For NXT/EV3 Touch → Pressed تنظیم می‌کنیم. در ادامه موتورها را متوقف کرده و سپس چرخش بر روی یک چرخ را انجام می‌دهیم:



شکل ۶۰ تشخیص برخورد دکمه‌ی سنسور با بلوک Wait For و چرخش رو به عقب ربات و تکرار این روند

به این ترتیب ربات مسیر مستقیم را تا زمانی که به مانع برخورد نکرده است ادامه می‌دهد. در صورتی که ربات به مانع برخورد کند، ۹۰ درجه چرخیده و سپس دوباره به مسیر مستقیم ادامه می‌دهد.

سنسور اولتراسونیک

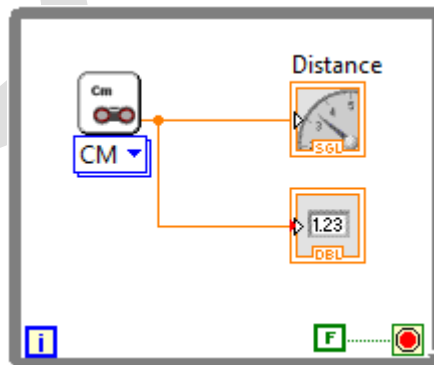


شکل ۶۱ سنسور اولتراسونیک مجموعه EV3

سنسور اولتراسونیک یا فراصوت، دارای یک فرستنده و یک گیرنده فراصوت می‌باشد که با کمک امواج فراصوت (امواج الکترومغناطیسی با فرکانس فراتر از محدوده شنوایی انسان) فاصله را تا مانع پیش رو، بدست می‌آورد. برای این عمل ابتدا فرستنده، یک پالس فراصوت را به سمت روبرو ارسال می‌کند. سپس پالس ارسال شده، بعد از رسیدن به مانع و بازتاب از آن، به سمت گیرنده بازمی‌گردد. با محاسبه‌ی زمان بین ارسال پالس و دریافت پالس بازتاب شده و همچنین دانستن سرعت صوت در هوا (برای هوای خشک در دمای ۲۰ درجه سلسیوس این مقدار برابر با 342.2 m/s می‌باشد)، می‌توان فاصله تا مانع را محاسبه نمود.

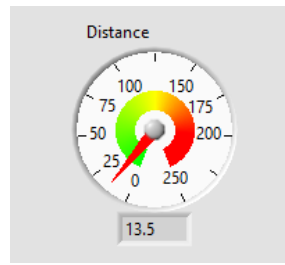
خواندن سنسور اولتراسونیک

برای خواندن سنسورها به طور کلی از بلوک Sensor در بخش I/O استفاده می‌کنیم. برای گرفتن اطلاعات از سنسور فراصوت و نمایش آن در نرم‌افزار لبرویو، می‌توان به چند طریق عمل کرد. ابتدا سنسور اولتراسونیک را توسط کابل به بریک EV3 متصل نموده (به هر یک از پورت‌های ۱ تا ۴) و بریک را به درگاه USB کامپیوتر وصل می‌کنیم. سپس برای خواندن اطلاعات سنسور و نمایش آن در لبرویو، بلوک Sensor را در حلقه بینهایت while قرار می‌دهیم تا به طور مداوم مقادیر خوانده شده از سنسور بروزسانی و نمایش داده شود:



شکل ۶۲ نحوه خواندن اطلاعات سنسور فراصوت

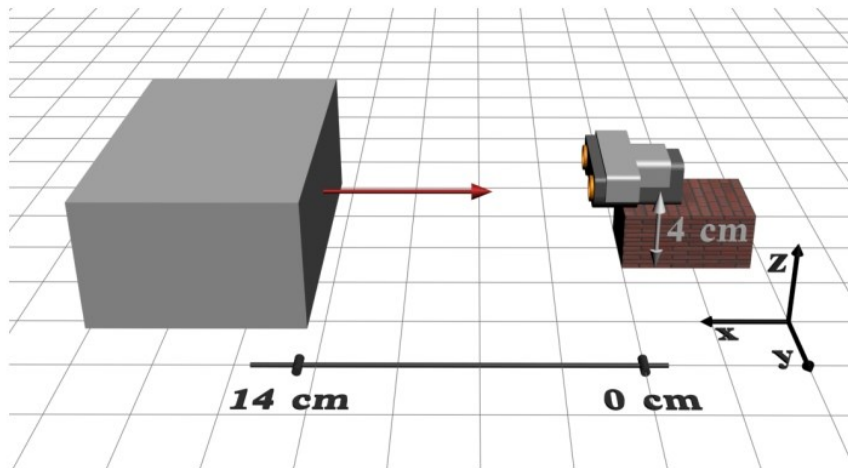
در قسمت Front Panel مقادیر خوانده شده بر حسب سانتی‌متر به شکل زیر نمایش داده می‌شود:



شکل ۶۳ مقدار خوانده شده از سنسور در Front Panel

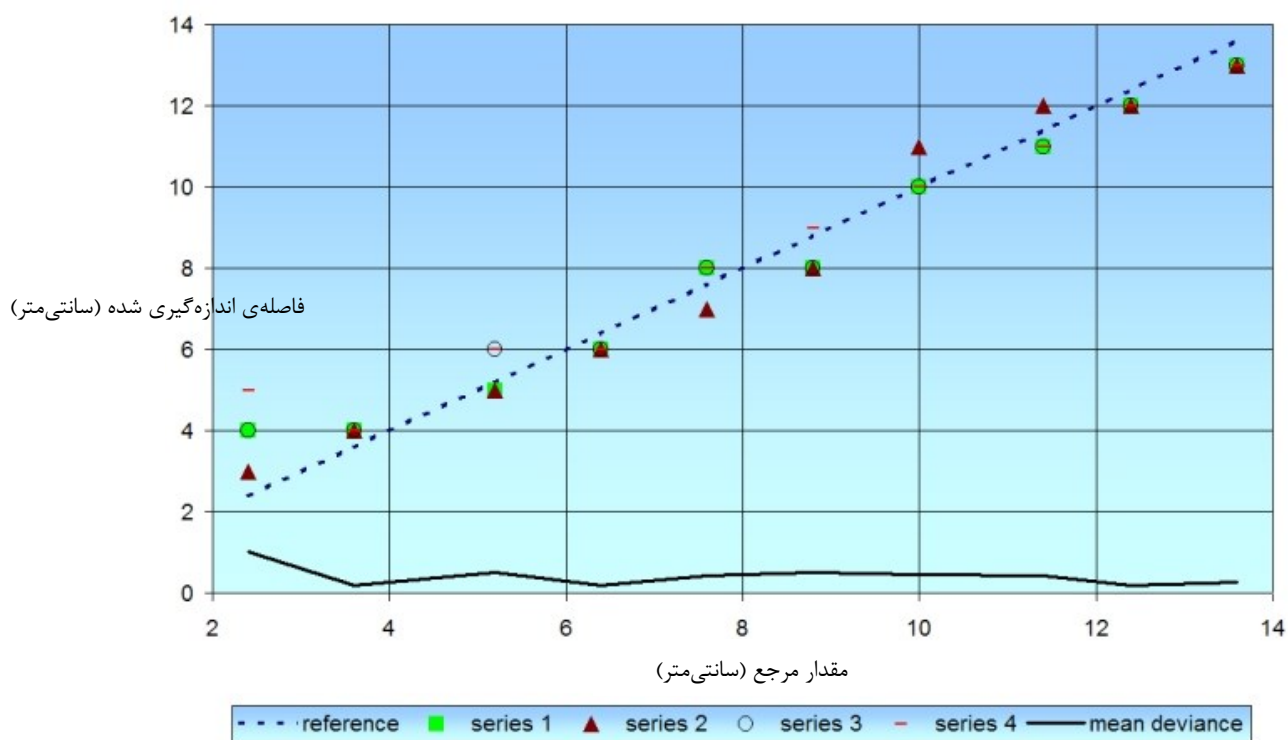
کالیبراسیون سنسور اولتراسونیک

برای بدست آوردن درکی از دقت سنسور اولتراسونیک و همچنین محیط پوشش داده شده توسط این سنسور، می توان آزمایش هایی را انجام داد. در شکل ۶۴ چیدمان مورد نیاز برای آزمایش نشان داده شده است:



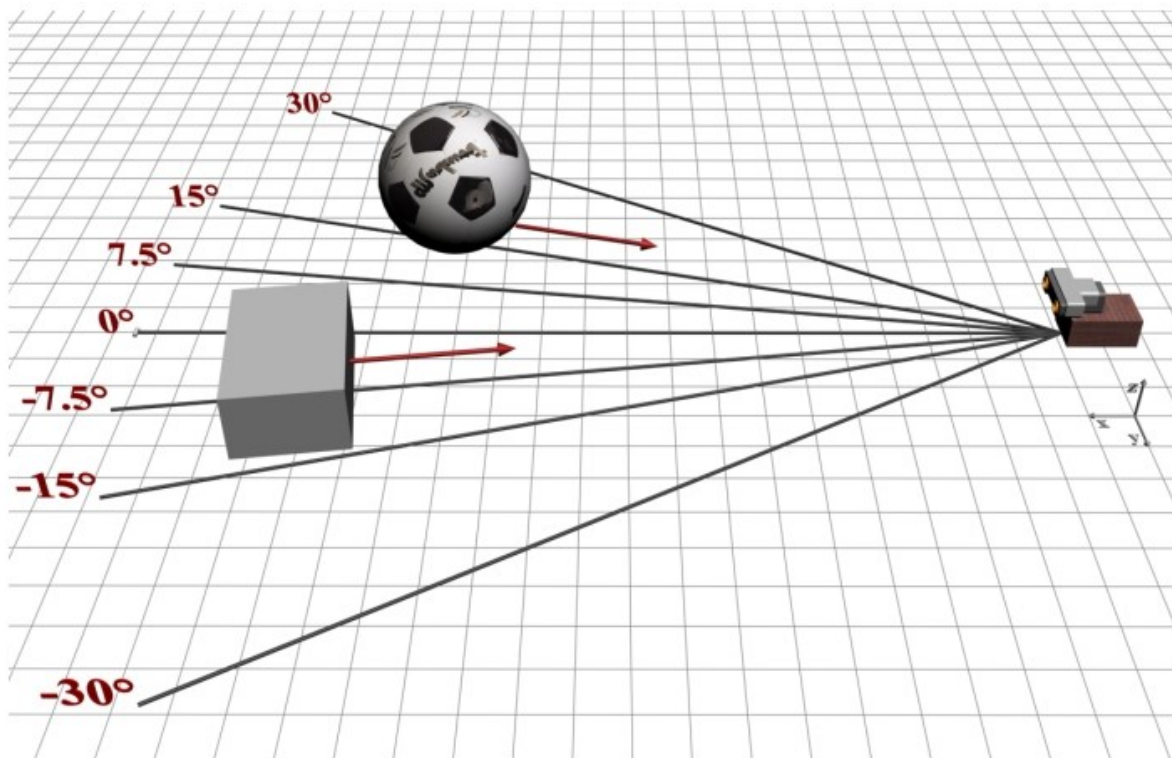
شکل ۶۴ موقعیت سنسور و مانع در آزمایش اول

با در نظر گرفتن مانع در روبروی سنسور و افزایش فاصله‌ی آن با سنسور، می‌توان نمودار شکل ۶۵ را رسم کرد که فاصله اندازه‌گیری شده توسط سنسور بر حسب فاصله‌ی واقعی را نشان می‌دهد. همچنین می‌توان مقدار انحراف میانگین از مقدار واقعی را رسم نمود.



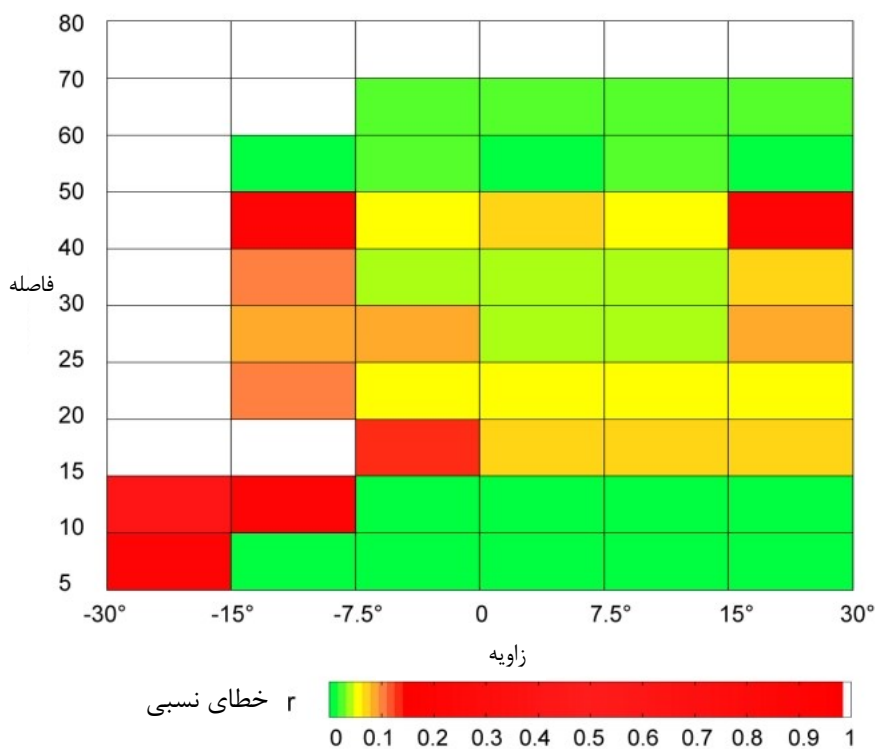
شکل ۶۵ نمودار فاصله اندازه‌گیری شده توسط سنسور بر حسب فاصله‌ی واقعی

فواصل کمتر از ۳ سانتی‌متر توسط سنسور قابل اندازه‌گیری نمی‌باشند. بیشترین انحراف از مقدار واقعی برابر با ۲.۶ سانتی‌متر در فاصله ۲.۵ سانتی‌متری می‌باشد. انحراف میانگین این سنسور برابر ۰.۴۰۸ سانتی‌متر (کمتر از ۰.۵ سانتی‌متر) است که نشان دهنده رفتار صحیح سنسور می‌باشد. در آزمایش دوم هدف ما بدست آوردن گستره‌ی دید این سنسور است. در این آزمایش نیز از همان مانع قبل استفاده گردیده است (۱۴.۵ در ۹.۵ در ۶ سانتی‌متر). در این حالت مانع را در فواصل و زوایای مختلف نسبت به سنسور جابجا کرده و فواصل خوانده شده را یادداشت می‌نماییم. سنسور در دو موقعیت افقی و عمودی قرار داده شده است. در شکل ۶۶ چیدمان با موقعیت افقی سنسور آورده شده است.



شکل ۶۶ بدست آوردن گستره‌ی دید در آزمایش دوم

در نمودار شکل ۶۷، نمایش گرافیکی از گستره‌ی دید در وضعیت افقی رسم گردیده است.

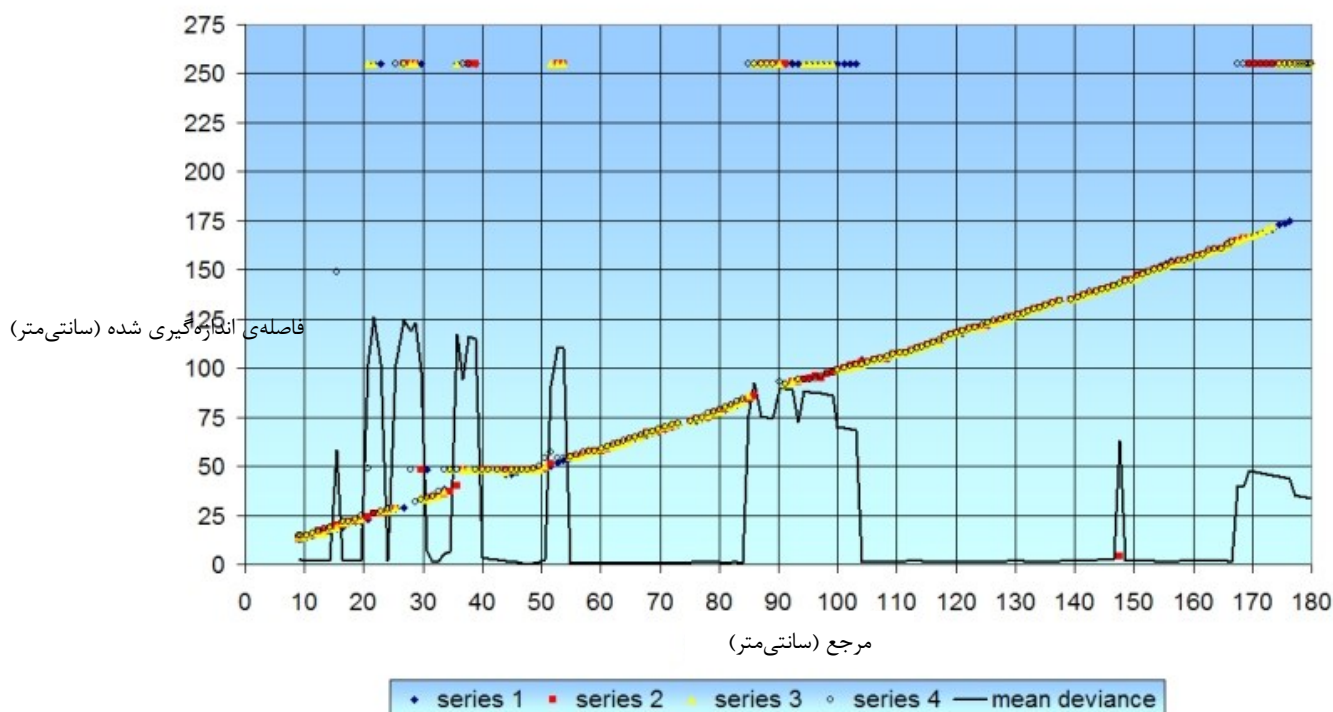


شکل ۶۷ نمودار زاویه دید افقی سنسور بر حسب فاصله مانع

نتایج نشان می‌دهند که سنسور اولتراسونیک همواره باید در وضعیت افقی قرار بگیرد زیرا در سایر وضعیت‌ها گستره و عمق دید کاهش می‌یابد. به نظر می‌رسد که سنسور مقداری در چشم‌چپ، ضعف بینایی دارد که

این مسئله به خاطر این است که چشم چپ این سنسور در واقع گیرنده‌ی امواج اولتراسونیک و چشم راست، فرستنده‌ی آن امواج است.

بعد از تست‌های فوق که برای رفتارهای آماری سنسور اولتراسونیک انجام گردید، اقدام به آزمایش‌های دینامیکی می‌نماییم. نمودار شکل ۶۸ فواصل خوانده شده توسط سنسور اولتراسونیک در هنگام نزدیک شدن به دیوار را نشان می‌دهد. اطلاعات این نمودار توسط برنامه‌ای که در نرم‌افزار LEGO تحت لیبویو رسم شده است که در آن مقادیر کنونی اولتراسونیک و مقادیر زاویه یکی از موتورهای متحرک در یک فایل در بریک ذخیره گردیده است. سپس این فایل از روی بریک بارگیری شده است.

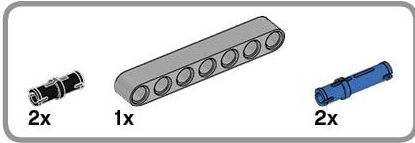


شکل ۶۸ نمودار نتیجه‌ی آزمون دینامیکی سنسور اولتراسونیک

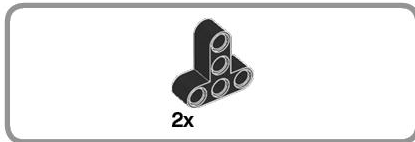
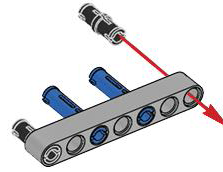
آزمون دینامیکی دو ضعف سنسور اولتراسونیک را آشکار می‌کند. مورد اول این است که این سنسور در بعضی نواحی به جای نشان دادن فاصله‌ی واقعی، مقدار ۲۵۵ سانتی‌متر را نمایش می‌دهد. مورد دوم که حتی اهمیت بیشتری دارد، ناحیه‌ی بحرانی بین ۲۵ سانتی‌متر و ۵۰ سانتی‌متر می‌باشد که سنسور در اکثر مواقع مقدار غلط ۴۸ سانتی‌متر را نمایش می‌دهد.

کاربردهای سنسور اولتراسونیک

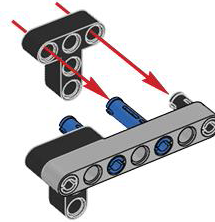
از این سنسور برای کاربردهای متفاوتی می‌توان بهره برد. یکی از کاربردهای آن برای جلوگیری از برخورد ربات با موانع می‌باشد. در ادامه ربات educator را به گونه‌ای اصلاح می‌کنیم تا موانع پیش روی خود را تشخیص داده و با تغییر مسیر، از برخورد با آنها جلوگیری کند. اصلاحات زیر را بر روی ربات educator انجام دهید:



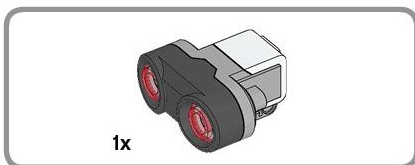
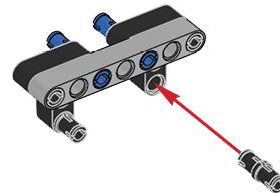
1



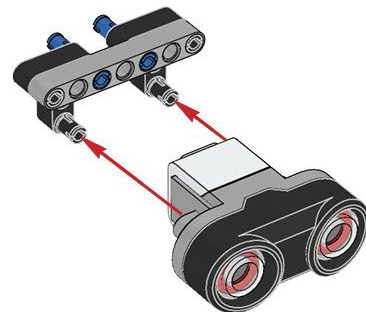
2



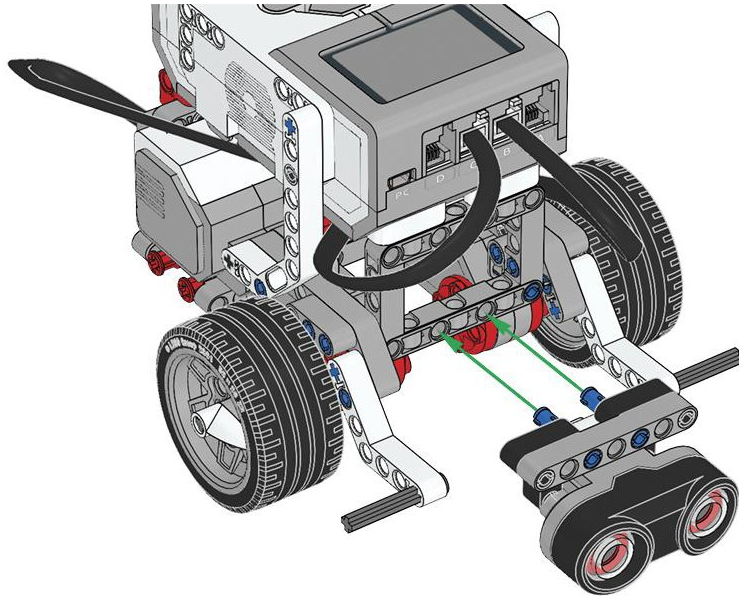
3



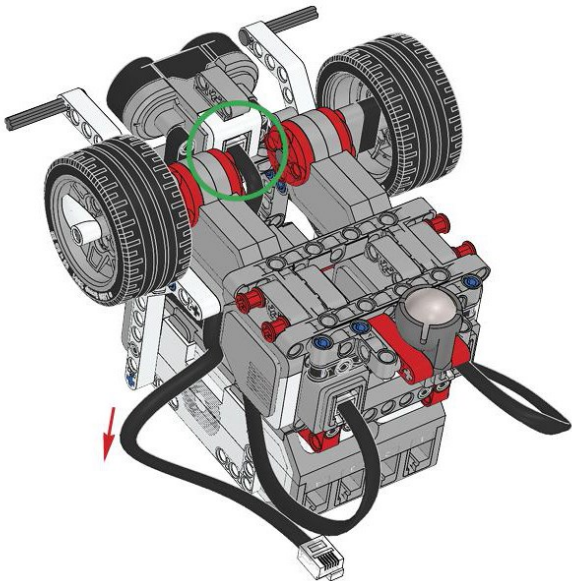
4



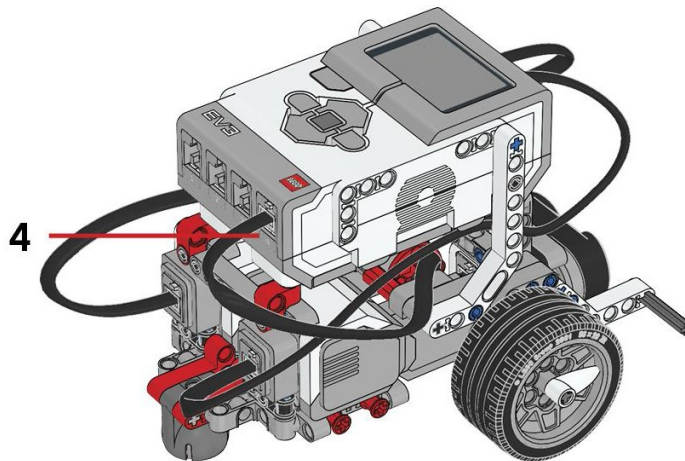
5

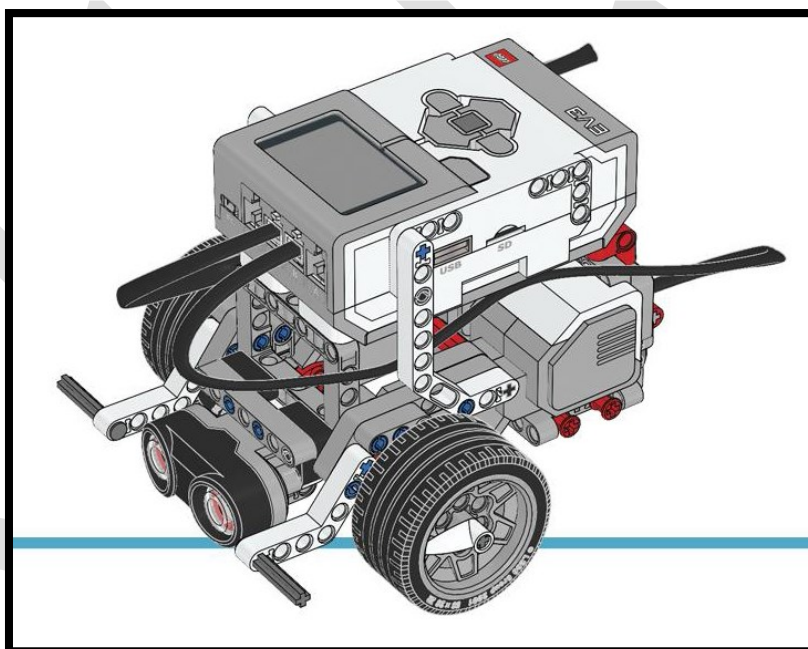
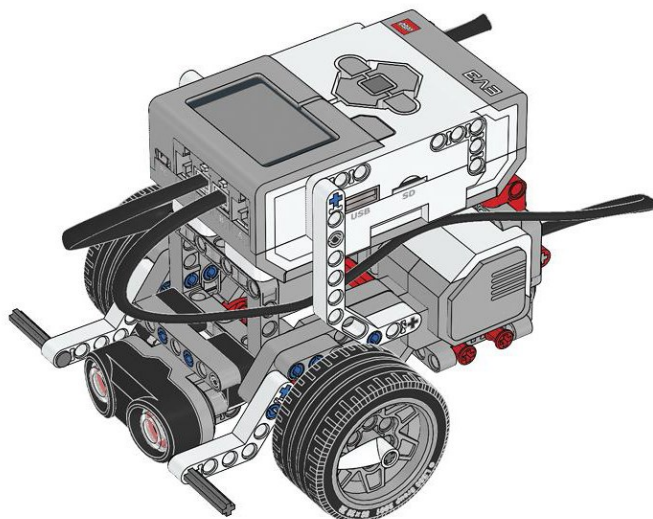


6



7





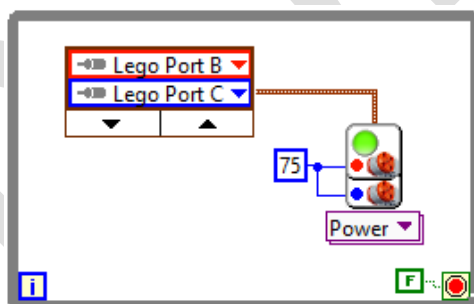
شکل ۶۹ نصب سنسور اولتراسونیک بر روی ربات educator

می‌خواهیم ربات را به گونه ای برنامه ریزی کنیم در حالت عادی ربات حرکت در خط مستقیم را انجام دهد و هنگامی که به نزدیکی یک مانع رسید چرخشی انجام داده و دوباره حرکت مستقیم خود را ادامه دهد، در اینصورت این ربات قادر خواهد بود تا همواره از برخورد به موانع جلوگیری کند.



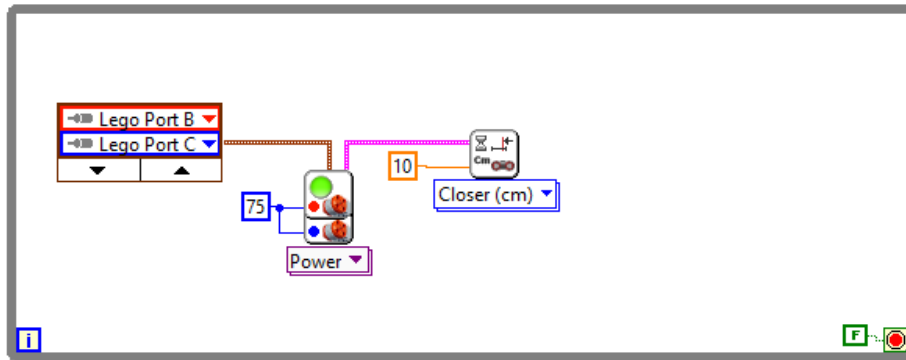
شکل ۷۰ تشخیص موانع با سنسور اولتراسونیک

پس ابتدا مانند زیر، حلقه‌ی بینهایت را برای تکرار همیشگی این روند قرار می‌دهیم و در آن حرکت مستقیم ربات را برنامه‌ریزی می‌کنیم:



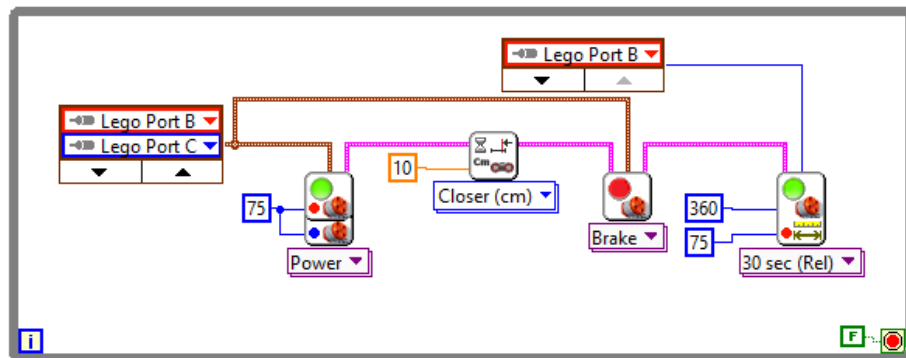
شکل ۷۱ حرکت موتورها در حلقه بینهایت while

در قدم بعد باید برای توقف موتورها و در نتیجه جلوگیری از برخورد با موانع، از بلوک Wait For استفاده کنیم. این بلوک دارای شروط دلخواهی است که با مشخص کردن آن، هنگام برقرار شدن شرط، فعالیت‌های ربات متوقف شده و فعالیت بعدی را انجام می‌دهد. در اینجا ما می‌خواهیم تا هنگامی که مانعی در کمتر از ۱۰ سانتی‌متری ربات شناسایی نشود، ربات به حرکت مستقیم خود ادامه دهد، و در صورت شناسایی مانع در فاصله کمتر از ۱۰ سانتی‌متر، ربات توقف کرده و به اندازه ۹۰ درجه در جهت ساعتگرد بچرخد. پس شرط بلوک Wait For را بر روی $Ultrasonic \rightarrow less\ than(cm)$ قرار داده و عدد ثابت ۱۰ را به عنوان ورودی به بلوک می‌دهیم:



شکل ۷۳ اضافه کردن بلوک wait for

عملی که بعد از Wait For باید انجام داد، ابتدا متوقف کردن چرخ‌ها و سپس حرکت یکی از چرخ‌ها به اندازه‌ی مناسب برای چرخش ۹۰ درجه ربات می‌باشد:



شکل ۷۲ اضافه کردن بلوک Stop Motors و چرخش ربات

سنسور ژيروسکوپ



شکل ۷۴ سنسور ژيروسکوپ مجموعه‌ی EV3

سنسور ژيروسکوپ به منظور اندازه‌گیری مقدار چرخش و یا اندازه‌گیری سرعت چرخش می‌باشد. سنسور ژيروسکوپ موجود در این مجموعه می‌تواند تنها چرخش حول یک محور را اندازه‌گیری نماید. سنسور ژيروسکوپ EV3، با بهره از تکنولوژی MEMS^۳ ساخته شده است. طرز کار سنسور ژيروسکوپ بر اساس نیروی ناشی از شتاب کریولیس است. جرم کوچکی از کریستال را داخل محفظه‌ای فلزی در نظر بگیرید، این جرم کوچک کریستال در هنگام اعمال ولتاژ الکتریکی مقداری منقبض می‌شود و در هنگام حذف ولتاژ، به اندازه اولیه خود باز می‌گردد. حال اگر به این جرم یک فنر متصل کنیم و جریان الکتریکی را دائماً قطع و وصل نماییم، جرم در راستای یک خط مستقیم نوسان می‌کند. همانطور که از قانون پایستگی انرژی می‌دانیم، اجسام متحرک مایل به حفظ حرکت در راستای حرکت خود می‌باشند مگر اینکه یک نیروی خارجی در جهت دیگر به آنها اعمال شود. هنگامی که سنسور می‌چرخد، جرم کریستالی حرکت خطی خود را در راستای اولیه نوسان، حفظ می‌کند اما به علت اینکه بدنه‌ی سنسور نیز چرخیده است، این جرم کریستالی دیگر هم راستا با بدنه‌ی سنسور نوسان نمی‌کند. سنسور ژيروسکوپ مقدار این انحراف را اندازه گرفته و مقدار چرخش را محاسبه می‌کند و بدین صورت می‌تواند مقدار چرخش و نرخ چرخش (سرعت زاویه‌ای) را اندازه‌گیری کند. این سنسور در بسیاری از صنایع استفاده می‌شود، از تلفن‌های همراه گرفته تا ماهواره‌ها و ربات‌های عظیم صنعتی.

خواندن سنسور ژيروسکوپ

ابتدا بلوک Sensor از بخش I/O → MINDSTORMS Robotics → functions را به بلوک دیاگرام خود اضافه می‌کنیم.

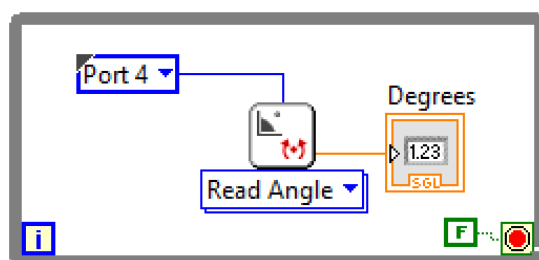


Touch_Pressed ▾

شکل ۷۵ بلوک Sensor

^۳ Micro Electro-Mechanical System

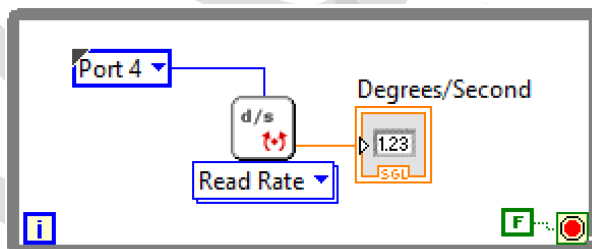
برای این سنسور ۲ مد یا حالت کاری تعریف شده است. یکی برای خواندن مقدار چرخش Read Angle و دیگری برای خواندن نرخ چرخش یا همان سرعت زاویه‌ای Read Rate. ابتدا این بلوک را بر روی Read Angle قرار داده و آنرا داخل حلقه‌ی بینهایت قرار می‌دهیم:



شکل ۷۶ خواندن مقدار زاویه از مرجع

در این حالت در هر لحظه مقدار زاویه‌ی اندازه‌گیری شده توسط سنسور ژيروسکوپ خوانده می‌شود و در Front Panel نمایش داده می‌شود. دقت شود که در این حالت زاویه‌ی خوانده شده نسبت به مرجع پیش‌فرض سنسور اندازه‌گیری می‌شود، در قسمت بعد که با نام کالیبراسیون سنسور ژيروسکوپ آمده است به تغییر این مرجع می‌پردازیم.

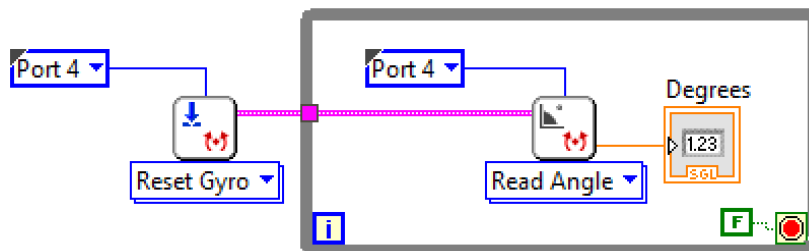
در مد دیگر این سنسور می‌توانیم نرخ تغییر زاویه را بخوانیم. مانند مد قبل سنسور را داخل حلقه‌ی بینهایت قرار داده و سپس گزینه‌ی Read Rate را انتخاب می‌کنیم. حال با اجرای برنامه، در هر لحظه می‌توان مقدار چرخش سنسور را در Front Panel مشاهده کرد:



شکل ۷۷ خواندن نرخ تغییر زاویه

کالیبراسیون سنسور ژيروسکوپ

همانطور که در قسمت قبل گفته شد، زاویه‌ی خوانده شده توسط سنسور نسبت به مرجع پیش‌فرض سنسور اعلام می‌شود. برای تغییر این مرجع می‌توان از بلوک Sensor و گزینه‌ی Reset Gyro استفاده کرد تا هر بار که برنامه را اجرا می‌کنیم مرجع همان زاویه‌ی اولیه باشد و سنسور نسبت به آن زاویه، سایر زوایا را بخواند:

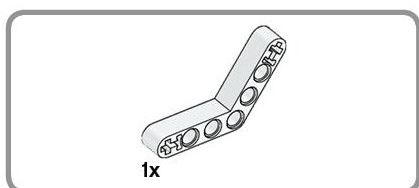
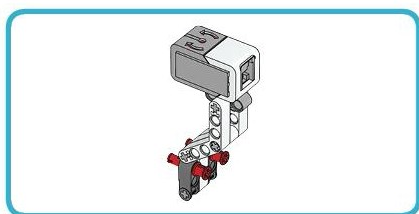


شکل ۷۸ ریست کردن زاویه مرجع و خواندن زوایا نسبت به زاویه‌ی اولیه‌ی سنسور

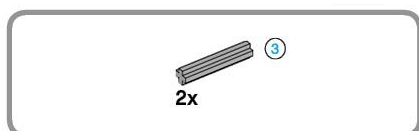
با اجرای برنامه‌ی بالا، ابتدا با ریست کردن سنسور ژيروسکوپ، زاویه‌ی مرجع به زاویه‌ی اولیه‌ی اجرای برنامه تغییر کرده و سپس سایر زوایای خوانده شده، نسبت به آن زاویه‌ی اولیه اعلام می‌شود.

کاربردهای سنسور ژيروسکوپ

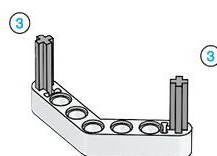
سنسور ژيروسکوپ در طیف گسترده‌ای از ربات‌ها مورد استفاده قرار می‌گیرد. ابتدا برای آشنایی بهتر با این سنسور، با استفاده از ربات پایه، برنامه‌ای می‌نویسیم که همواره جهت ربات را به یک سو حفظ کند و ربات در مقابل تغییر جهت مقاومت نماید. همانطور که در ربات تعقیب خط با کنترلر تناسبی آشنا شدیم، در اینجا نیز از این نوع کنترل برای پیاده‌سازی این هدف استفاده می‌کنیم. برای این منظور ابتدا سنسور ژيروسکوپ را بر روی ربات Educator متصل می‌کنیم:

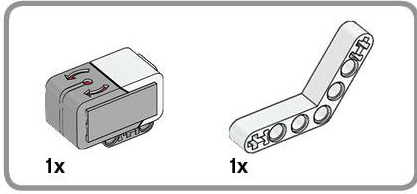


1

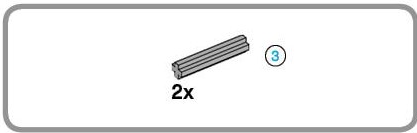
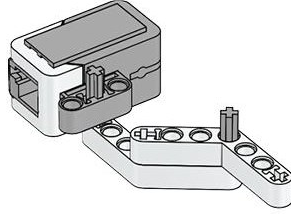


2

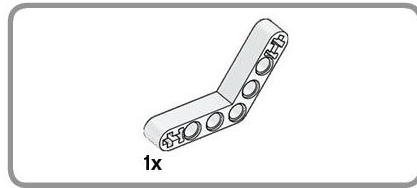
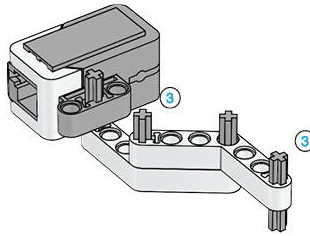




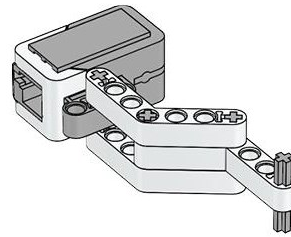
3

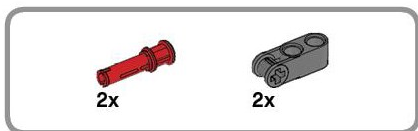


4

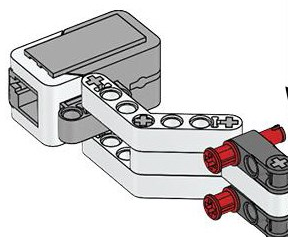
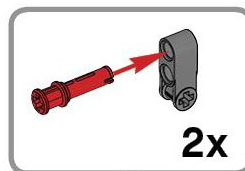


5

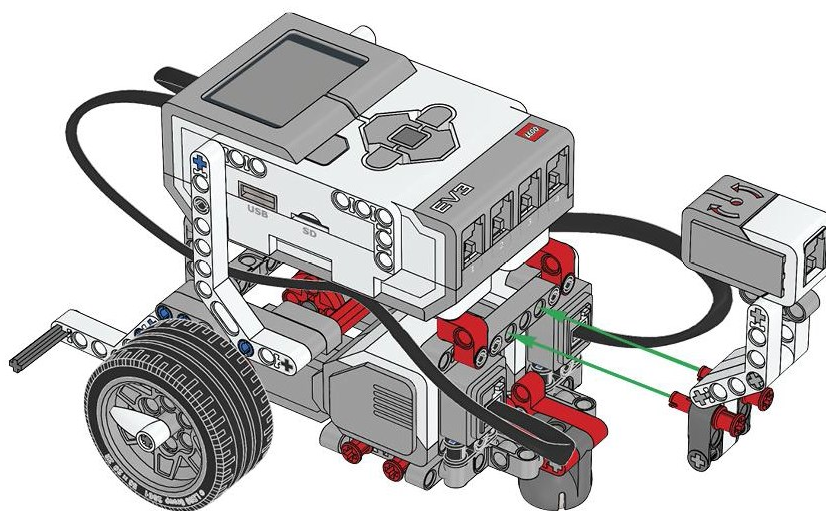




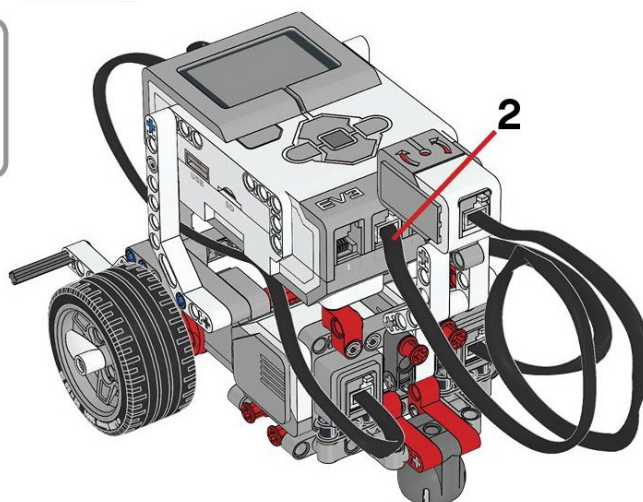
6



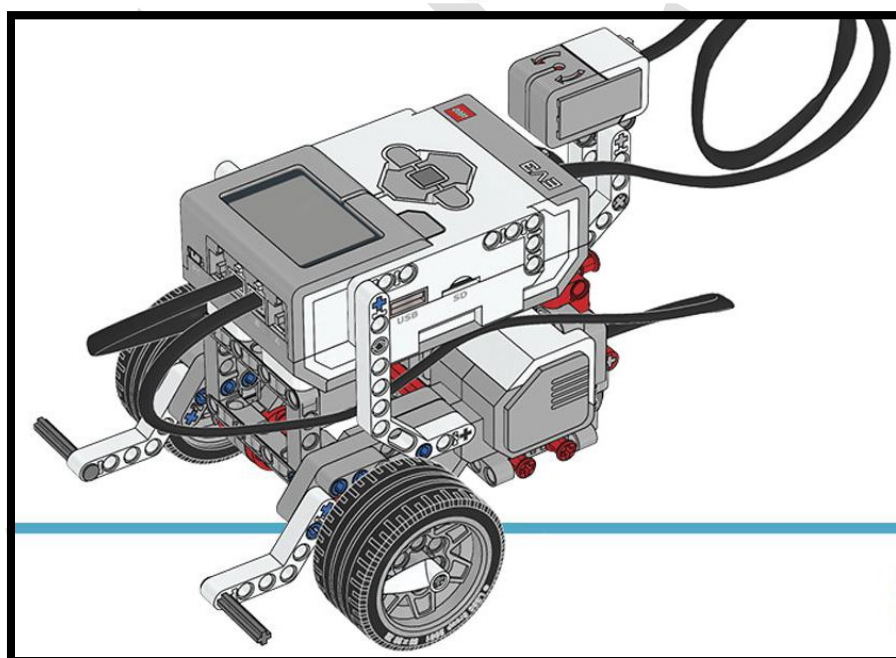
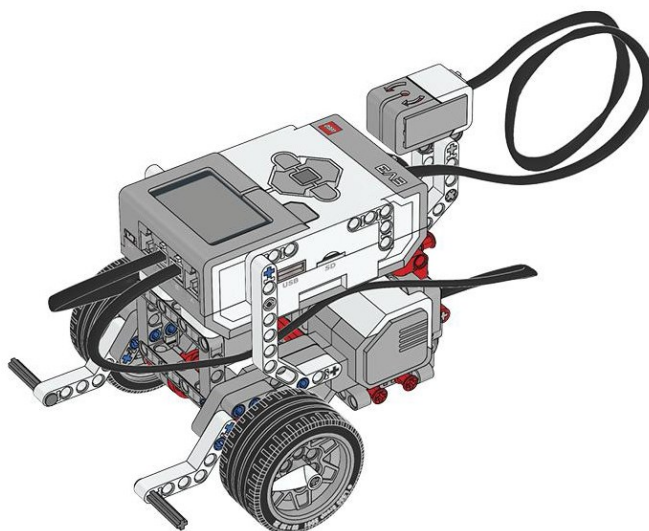
7



8

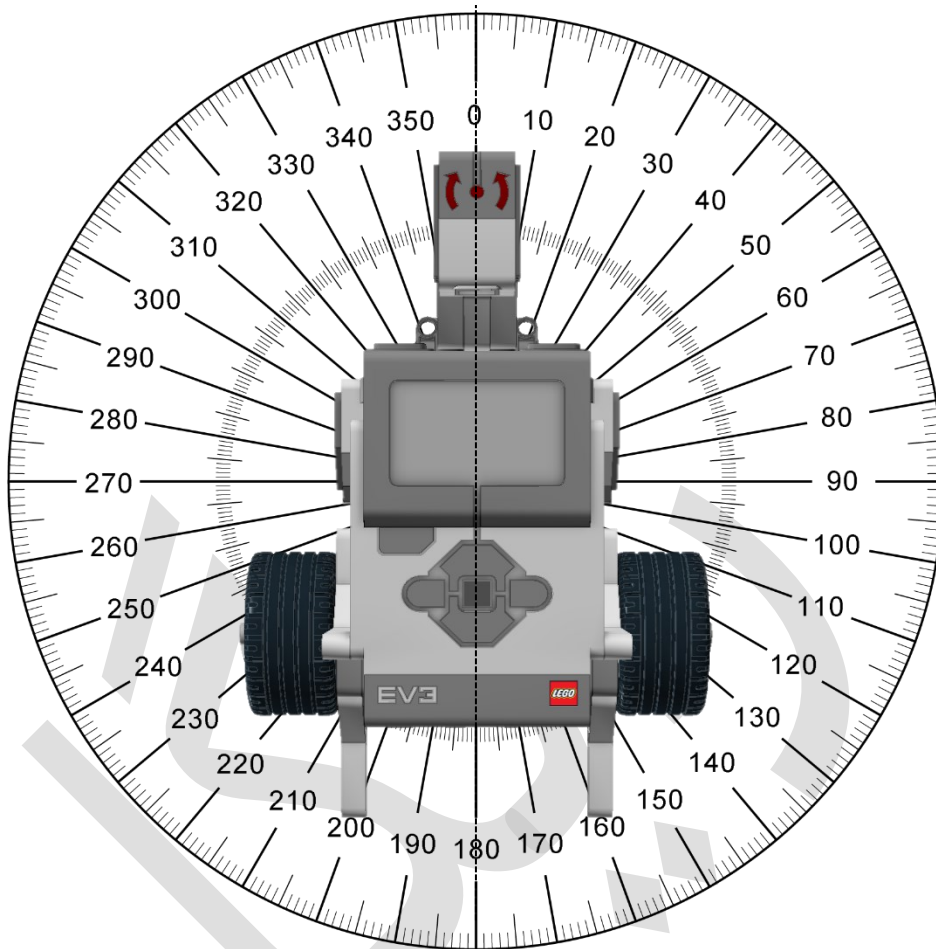


9



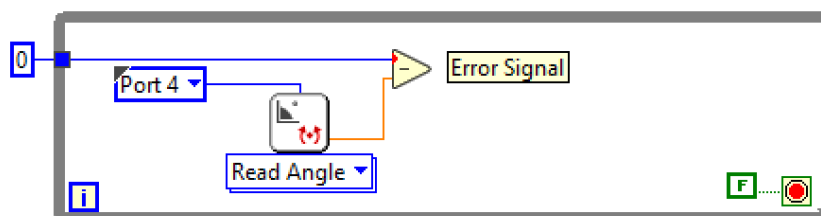
شکل ۷۹ اتصال سنسور ژيروسکوپ به ربات Educator

فرض کنید می‌خواهیم همواره ربات در زاویه‌ی صفر درجه قرار بگیرد، پس مقدار مرجع ما صفر است.



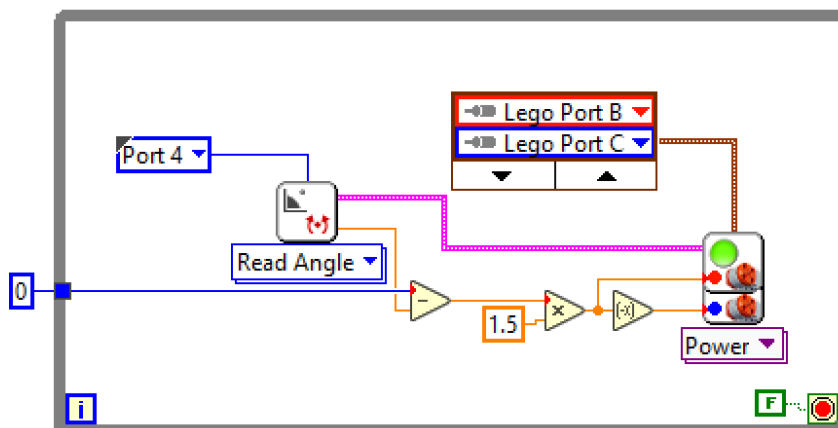
شکل ۸۰ زاویه‌ی اندازه‌گیری شده توسط سنسور ژيروسکوپ

حال باید سیگنال خطا را ایجاد نماییم و بعد از ضرب یک ضریب تقویت در آن، سیگنال تقویت شده را به موتورها اعمال کنیم. برای ساخت سیگنال خطا به صورت زیر عمل می‌نماییم:



شکل ۸۱ تشکیل سیگنال خطا

سپس این سیگنال را در یک عدد ثابت ضرب کرده و به موتورها اعمال می‌کنیم. دقت کنید که هدف ما در اینجا چرخش ربات حول محور خودش می‌باشد، پس باید جهت چرخش موتورها عکس یکدیگر باشد، پس باید سیگنال تقویت شده را به یک موتور و منفی آن سیگنال را به موتور دیگر اعمال کرد.



شکل ۸۲ تنظیم موتور بر روی زاویه‌ی صفر درجه با چرخش چرخها

در نهایت برنامه‌ی مورد نظر به شکل بالا درمی‌آید. حال با اجرای این برنامه، ربات همواره جهت خود را به سمت زاویه‌ی صفر حفظ می‌کند و در صورت تغییر جهت، متناسب با اختلافی که از زاویه‌ی صفر دارد، چرخش کرده و خود را به زاویه‌ی صفر می‌رساند.

سنسور رنگ/نور



شکل ۸۳ سنسور رنگ مجموعه EV3

امواج مرئی شامل بخش کوچکی از طیف امواج الکترومغناطیسی از طول موج ۴۰۰ تا ۷۰۰ نانومتر می‌باشد. با تغییر طول موج در امواج مرئی، رنگ طیف تابیده شده تغییر می‌یابد. استفاده از سنسور رنگ در رباتیک و صنعت کاربردهای بسیاری دارد. برای شناسایی رنگ طیف دریافت شده، در ساده‌ترین حالت می‌توان از فوتورزیستورها استفاده کرد که با تغییر رنگ، مقاومت الکتریکی مختلفی را اعمال می‌کند. اما پرکاربردترین سنسورهای رنگ، توسط تکنولوژی نیمه رسانا ساخته شده است که در این حالت LEDهایی وجود دارد که با توجه به هر رنگ، جریان الکتریکی متفاوتی را ایجاد می‌کنند. معمولاً برای شناسایی دقیق‌تر رنگ هر جسم، از LEDهایی برای تاباندن نور سفید به جسم استفاده می‌شود. سپس با اعمال فیلترهای مشخص بر روی طول موج امواج دریافت شده، می‌توان آنرا به سه رنگ قرمز، سبز و آبی تجزیه نمود. این سه رنگ، از رنگ‌های اصلی می‌باشند که از ترکیب آنها می‌توان برای تولید سایر رنگ‌ها استفاده کرد. اساس کار چشم انسان نیز به همین شکل می‌باشد.

سنسور موجود در مجموعه EV3 علاوه بر قابلیت شناسایی مقادیر سه رنگ اصلی قرمز، سبز و آبی، قابلیت شناسایی عدم وجود رنگ (clear) و شدت نور را دارد. در ادامه روش استفاده از این سنسور برای اهداف مختلف را بررسی می‌کنیم.

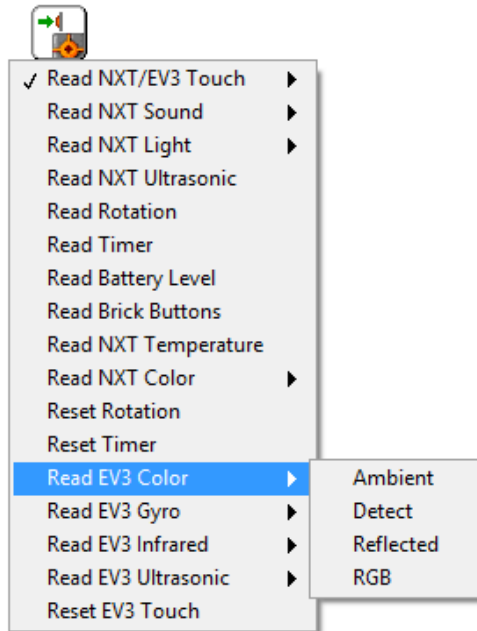
خواندن سنسور رنگ/نور

همانطور که برای خواندن مقادیر سایر سنسورها از بلوک Sensor در بخش MINDSTORMS → functions I/O → Robotics استفاده کردیم، برای خواندن مقادیر این سنسور نیز به روش مشابه، این بلوک را به بلوک دیگرام برنامه خود اضافه می‌نماییم.



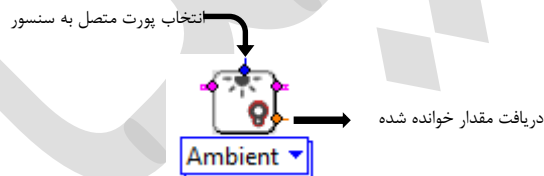
شکل ۸۴ بلوک Sensor

بعد از قرار دادن بلوک sensor، با کلیک بر روی نوار پایین بلوک، به شاخه Read EV3 Color می‌رویم. در اینجا ۴ گزینه طبق شکل زیر وجود دارد که در ادامه به ترتیب آنها را شرح می‌دهیم:



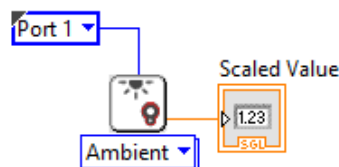
شکل ۸۵ حالت‌های مختلف برای سنسور رنگ

- مد اول Ambient می‌باشد، به این منظور که در این حالت سنسور در مد اندازه‌گیری شدت نور محیط می‌باشد. با انتخاب این حالت در بلوک یک گرهی ورودی و یک گرهی خروجی ایجاد می‌شود:



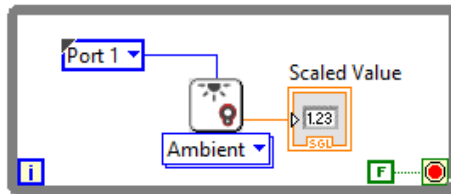
شکل ۸۶ ورودی و خروجی‌های سنسور رنگ

برای مشخص کردن پورتهی که سنسور به آن متصل است بر روی گرهی بالای بلوک کلیک راست کرده و گزینهی **Create → Constant** را انتخاب می‌کنیم و از نوار ایجاد شده، پورت مناسب را انتخاب می‌کنیم. در قدم بعد برای نشان دادن مقدار دریافت شده که مبین مقدار نور موجود در محیط می‌باشد، بر روی گرهی سمت راست بلوک کلیک راست کرده و گزینهی **Create → Indicator** را کلیک می‌کنیم. در این حالت در **front panel** پنجره‌ای با عنوان **Scaled Value** ایجاد می‌شود که مقدار خوانده شده‌ی سنسور در آن نمایش داده می‌شود:



شکل ۸۷ اتصال ورودی و خروجی سنسور رنگ

برای اینکه این مقدار به طور مداوم تکرار شود، آنرا در یک حلقه‌ی **while** بینهایت قرار می‌دهیم:



شکل ۸۸ قرار دادن فرآیند اندازه‌گیری در حلقه‌ی بینهایت

در این حالت با آپلود برنامه بر روی بریک و اجرای مداوم برنامه، می‌توان مقدار خوانده شده سنسور را مشاهده کرد.

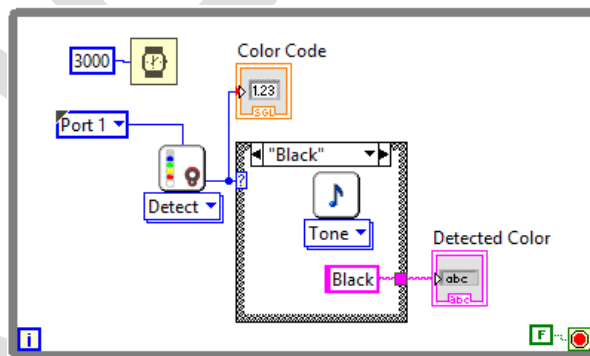
- مد بعدی سنسور Detect می‌باشد. در این مد، سنسور ۷ رنگ را شناسایی کرده و هر رنگ را با یک عدد مشخص می‌کند. با توجه به جدول زیر، هر عدد به یک رنگ نسبت داده شده است:

۱	مشکی	۵	قرمز
۲	آبی	۶	سفید
۳	سبز	۷	قهوه‌ای
۴	زرد	۰	عدم وجود رنگ

جدول ۲ کد متناظر با هر رنگ

این حالت نیز مانند حالت قبل، بلوک دارای یک گره برای مشخص کردن پورت متصل به سنسور و یک گره دیگر برای رنگ شناسایی شده می‌باشد.

برای خواندن سنسور رنگ و شناسایی رنگ‌ها، می‌توان به مشابه حالت قبل، سنسور را در یک حلقه‌ی بینهایت قرار داده و خروجی آنرا با توجه به جدول بالا نشان دهیم.

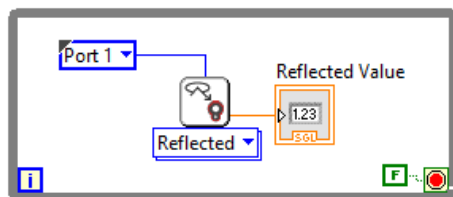


شکل ۸۹ خواندن و شناسایی رنگ توسط سنسور رنگ

با اضافه کردن یک Case Structure می‌توان رنگ شناسایی شده را بر روی Front Panel مشاهده کرد. بلوک Wait (ms) برای ایجاد فاصله‌ی زمانی بین هر بار شناسایی رنگ قرار داده شده است که باعث می‌شود هر ۳ ثانیه (۳۰۰۰ میلی‌ثانیه) یکبار، سنسور اقدام به شناسایی رنگ نماید.

- مد سوم سنسور به نام Reflected می‌باشد. در این حالت سنسور اقدام به اندازه‌گیری نور بازتابیده شده از سطح می‌نماید. این حالت کاربردهای فراوانی به خصوص برای طراحی ربات‌های تعقیب خط دارد. به این صورت که مقادیر خوانده شده از بازتاب خط مشکی با مقادیر خوانده شده از سطح زمین تفاوت دارد. مانند

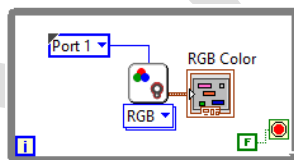
حالت‌های قبل در این حالت نیز پورت متصل به سنسور را مشخص کرده و مقدار خوانده شده را بر روی Front Panel نمایش می‌دهیم.



شکل ۹۰ اندازه‌گیری و نمایش مقدار بازتاب شده از سطح

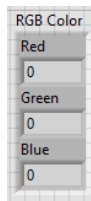
در بخش بعدی به کالیبراسیون این حالت از سنسور خواهیم پرداخت تا بتوانیم به درستی ربات تعقیب خط و سایر ربات‌های مشابه را طراحی کنیم.

- مد آخر سنسور رنگ/نور، RGB می‌باشد. RGB مخفف سه رنگ قرمز، سبز و آبی است که به عنوان رنگ‌های پایه شناخته می‌شوند. در این حالت سنسور رنگ شناسایی شده را به صورت ترکیبی از این سه رنگ نشان می‌دهد. با مشخص کردن پورت متصل به سنسور و همچنین ایجاد نشانگر (Indicator) با راست کلیک کردن بر روی گرهی RGB Color و انتخاب گزینه‌ی Indicator → Create می‌توان بلوک RGB Color را به مجموعه اضافه کرد.



شکل ۹۱ اندازه‌گیری و نمایش مقادیر RGB

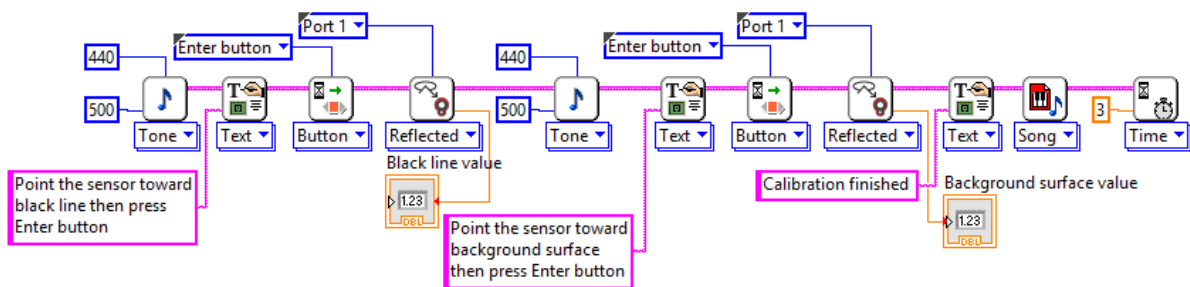
در این حالت در Front Panel سه مقدار RGB نشان داده می‌شود.



شکل ۹۲ نمایش مقادیر RGB در Front Panel

کالیبراسیون سنسور رنگ/نور

همانطور که گفته شد سنسور رنگ/نور قابلیت اندازه‌گیری شدت نور، شناسایی رنگ، مقادیر RGB و بازتاب را دارد. در سه حالت اول، سنسور با توجه به تنظیماتی که در کارخانه بر روی آن انجام گرفته است کار می‌کند و در اکثر موارد کالیبراسیون برای کاربردهای خاص لازم نمی‌باشد. اما در حالت اندازه‌گیری بازتاب، لازم است تا با توجه به شرایط و فضای کار ربات، آنرا تنظیم نموده و برای طراحی صحیح و دقت مناسب ربات، مقادیر خوانده شده بر روی خط مشکی و پس زمینه را به عنوان معیار مشخص کرد. برای این منظور برنامه‌ای به شکل زیر ایجاد می‌کنیم تا با انجام مراحل آن می‌توان مقادیر مناسب برای خط مشکی و پس زمینه را استخراج و اندازه‌گیری نمود.

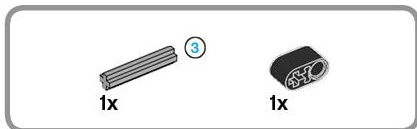
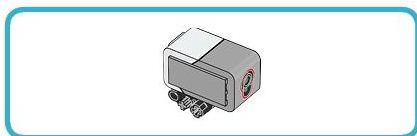


شکل ۹۳ فرآیند کالیبراسیون سنسور رنگ/نور

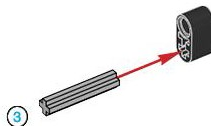
در ابتدا بوق کوتاهی برای اعلام شروع کالیبراسیون به صدا درآمده و سپس بر روی نمایشگر بزرگ پیامی مبتنی بر قرار دادن سنسور به سمت خط مشکی ایجاد می‌شود و دستگاه منتظر می‌ماند تا دکمه‌ی Enter بر روی بزرگ فشرده شود، در قسمت بعد مقدار خوانده شده را تحت عنوان Black line value در Front Panel نشان می‌دهد. مرحله‌ی بعد برای خواندن مقدار پس زمینه است که مشابه حالت قبل بوق و پیام بر روی بزرگ ایجاد می‌شود و نتیجه را تحت عنوان Background surface value نمایش داده می‌شود. در نهایت پیام "اتمام کالیبراسیون" و آهنگی پخش می‌شود که پایان یافتن کالیبراسیون را اعلام می‌کند. از مقادیر خوانده شده برای خط مشکی و پس زمینه در طراحی ربات تعقیب خط و ... می‌توان استفاده کرد. در بخش بعد از این مقادیر برای ساخت ربات تعقیب خط بهره می‌بریم.

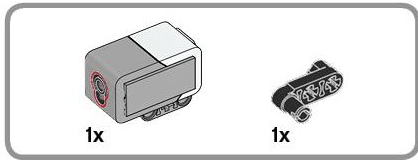
کاربردهای سنسور رنگ/نور

یکی از پرکاربردترین استفاده از سنسور نور مربوط به ربات‌های تعقیب خط می‌باشد. این نوع ربات‌ها در صنعت و در خطوط تولید مورد استفاده بسیار قرار گرفته‌اند. برای شروع با طراحی چنین رباتی، ابتدا لازم است تا تغییرات زیر را بر روی ربات پایه Educator انجام دهید:

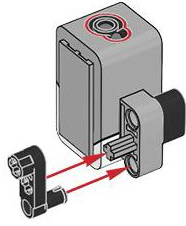


1

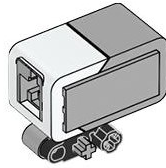




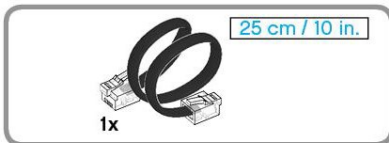
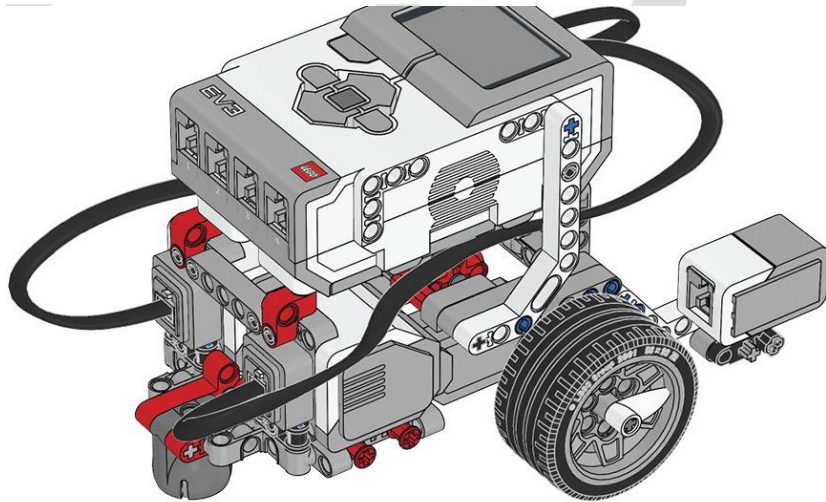
2



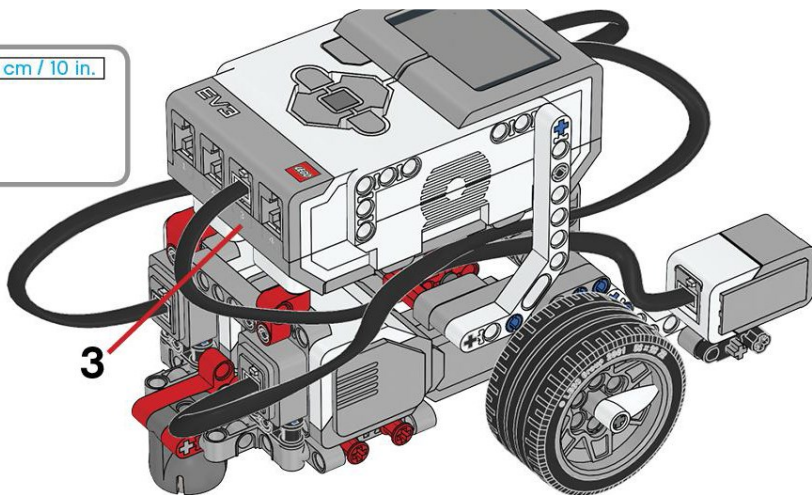
3

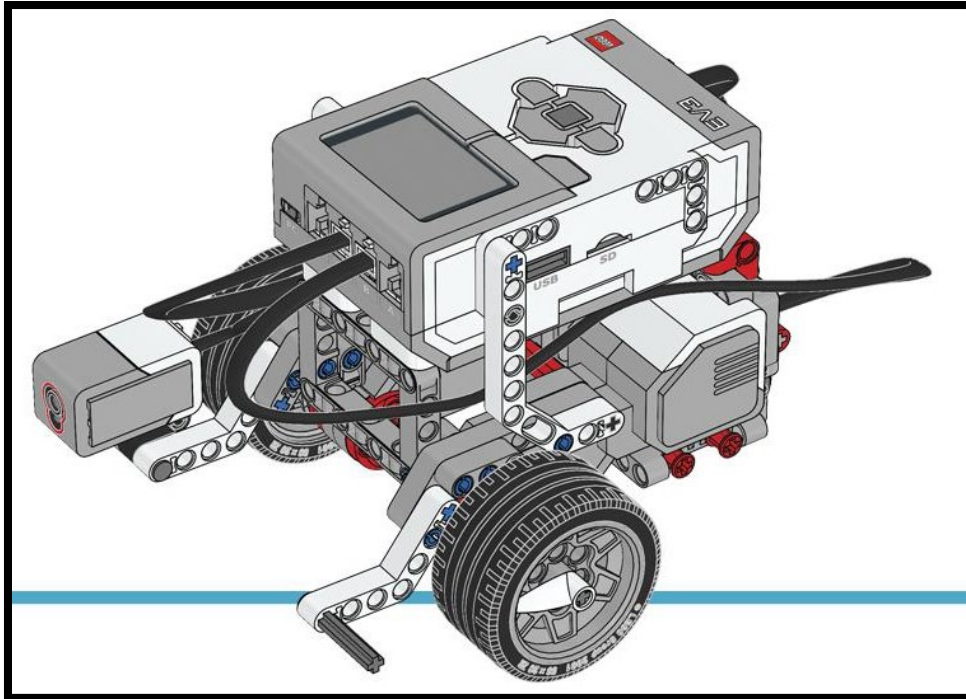


4



5





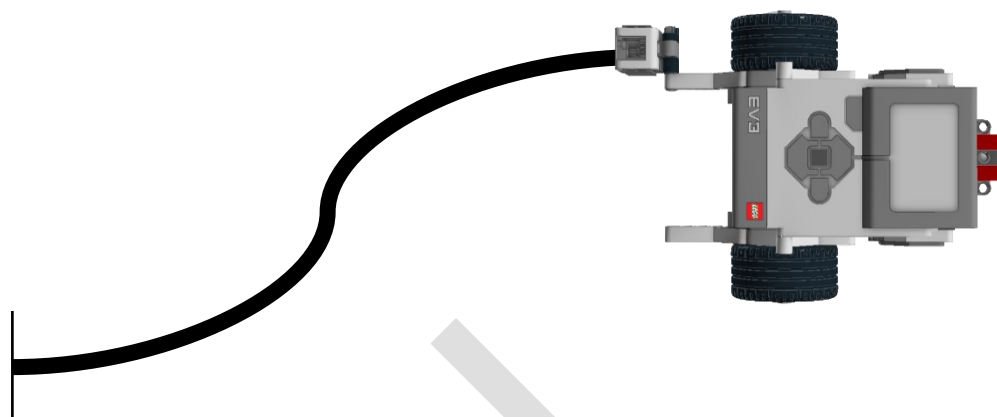
شکل ۹۴ نصب سنسور رنگ بر روی ربات Educator

برای ساخت یک ربات تعقیب خط لازم است تا ابتدا مسئله را به طور واضح تعریف کنیم. برای تعیین مسیر حرکت ربات یکی از راه ها، برنامه ریزی ربات و اندازه گیری مسیر مورد نظر است. اما در صورت تغییر مسیر، برنامه ی ربات باید بروز رسانی و اصلاح شود. علاوه بر اینکه معمولا در اندازه گیری و همچنین پیمودن مسافت های تعیین شده توسط ربات، خطاهایی وجود دارد که در صورت تکرار چندبار یک مسیر، ربات دیگر در مسیر مورد نظر قرار نداشته و یا دقت لازم را ندارد.

راه دیگری که برای تعیین مسیر و پیمایش آن برای ربات می توان در نظر گرفت، تعقیب خط تعبیه شده در مسیر مورد نظر است. در این حالت می توان مطمئن شد که ربات همیشه در مسیر مورد نظر حرکت می کند و هیچگاه دچار خطا یا کاهش دقت نخواهد شد.

ربات تعقیب خط باید بتواند خط کشیده شده بر روی زمین را از سطح پس زمینه تمییز دهد. در اینجا رباتی طراحی می کنیم که خط مشکی را از سطحی که ربات بر روی آن حرکت می کند تشخیص داده و آنرا دنبال کند. برخلاف اینکه این ربات، ربات تعقیب خط است اما در واقع ربات خط را دنبال نمی کند! بلکه هدف این ربات دنبال کردن لبه ی خط (مرز بین خط و زمینه) است. به علت اینکه در اینجا قرار است فقط از یک سنسور نور استفاده کنیم، در صورت دنبال کردن خط، با پیچش مسیر، ربات نمی تواند جهت پیچش را تشخیص دهد

و در نتیجه قادر به دنبال کردن خط نمی‌باشد. اما اگر هدف ما دنبال کردن لبه‌ی خط باشد، می‌توان جهت پیچش مسیر را تشخیص داد و متناسب با آن ربات را چرخاند.



شکل ۹۵ تعقیب خط توسط سنسور نور/رنگ ربات

برای تشخیص خط مشکی و پس زمینه، طبق بخش قبل که تحت عنوان کالیبراسیون ذکر شد، مقادیر خوانده شده‌ی سنسور برای بازتاب از خط مشکی و پس زمینه را بدست می‌آوریم. شدت بازتاب نور از خط مشکی بسیار کمتر از سطح زمینه‌ی روشن می‌باشد. هر چه به لبه‌ی خط مشکی نزدیک شویم، مقدار بازتاب افزایش یافته و با رسیدن به سطح زمینه، این مقدار به حداکثر مقدار خود می‌رسد.



شکل ۹۶ مقادیر خوانده شده بر روی خط و زمینه

در الگوریتم‌هایی که در ادامه به آنها می‌پردازیم، از این مقادیر برای طراحی و برنامه‌ریزی ربات استفاده می‌کنیم.

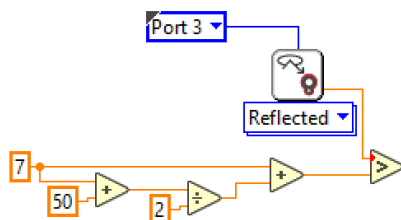
الگوریتم ساده

الگوریتم‌های متفاوتی را می‌توان برای این هدف در نظر گرفت. ساده‌ترین الگوریتم برای دستیابی به این هدف، استفاده از الگوریتم دو حالتی است. به این صورت که در هر لحظه، ربات مقدار اندازه‌گیری شده‌ی بازتاب را از سنسور خوانده و با مقایسه‌ی این مقدار با مقدار بازتاب در مرز خط مشکی (حد وسط مقدار خوانده شده بر روی خط مشکی و بر روی پس زمینه)، جهت چرخش ربات را مشخص می‌کنیم. در واقع در این الگوریتم، ربات ما یا در حال پیچش به راست است یا در حال پیچش به چپ. به این ترتیب ربات، قدم به قدم خط را دنبال می‌کند.

ابتدا قرار است که مقدار خروجی سنسور را خوانده و با مقدار بازتاب در مرز خط مقایسه کنیم. برای محاسبه‌ی مقدار بازتاب در مرز خط، می‌توان از رابطه‌ی زیر استفاده کرد:

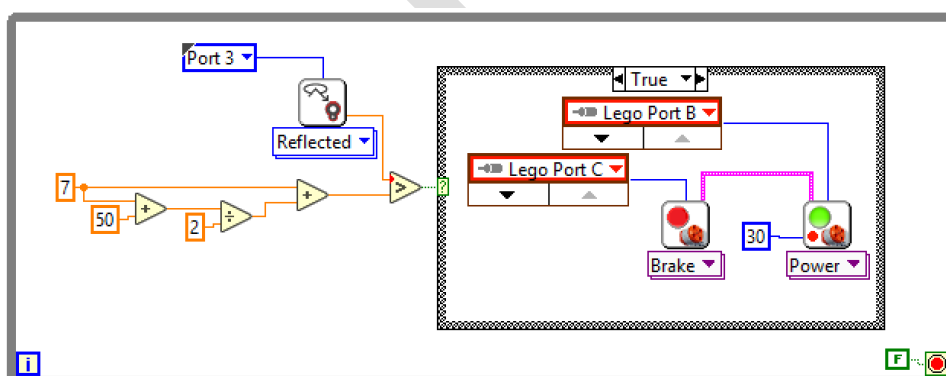
$$\text{بازتاب خط} = \frac{(\text{بازتاب خط} + \text{بازتاب زمینه})}{2} = \text{بازتاب روی مرز}$$

بلوک دیاگرام تا به اینجای کار به شکل زیر است:



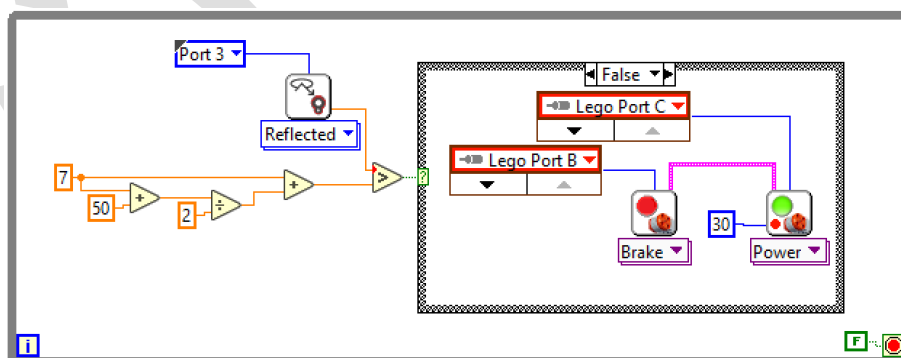
شکل ۹۷ بدست آوردن مقدار بازتاب روی مرز و مقایسه‌ی مقدار خروجی سنسور با آن

در ادامه قرار است که در صورت بیشتر بودن این مقدار از مقدار روی مرز، ربات به سمت خط مشکی بچرخد. اگر بخواهیم همواره لبه‌ی چپ مسیر را دنبال کنیم این چرخش به سمت راست خواهد بود، پس چرخ سمت راست را ثابت نگه داشته و چرخ سمت چپ را حرکت می‌دهیم. برای اینکه همیشه این فرآیند مقایسه‌ای تکرار شود باید کل این بلوک‌ها را درون حلقه While بینهایت قرار دهیم:



شکل ۹۸ چرخش به سمت راست برای نزدیک شدن به خط مشکی

برای عکس حالت گفته شده، باید موتورها را به صورت عکس فرمان داد. یعنی وقتی مقدار خوانده شده از مقدار روی مرز کمتر باشد (= بیشتر نباشد)، یعنی ربات به درون خط مشکی حرکت کرده است پس باید چرخ سمت چپ ثابت مانده و چرخ سمت راست حرکت کند تا ربات به سمت چپ متمایل شود.



شکل ۹۹ چرخش به سمت چپ برای فاصله گرفتن از خط مشکی

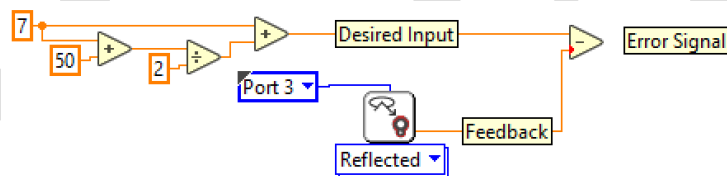
* دقت کنید که همیشه از بلوک Stop Motors برای ثابت نگه داشتن موتوری که قرار است ثابت باشد استفاده کنید در غیر اینصورت موتور همچنان فرمان قبلی را تکرار می‌کند.

الگوریتم تناسبی

در این حالت ما می‌خواهیم تا حرکت تعقیب خط ربات را نرمتر و یکنواخت تر کنیم. یعنی به جای اینکه ربات به طور مداوم در حال چرخش به چپ یا راست باشد، خط را دنبال کرده و به طور پیوسته جهت خود را نیز تغییر دهد.

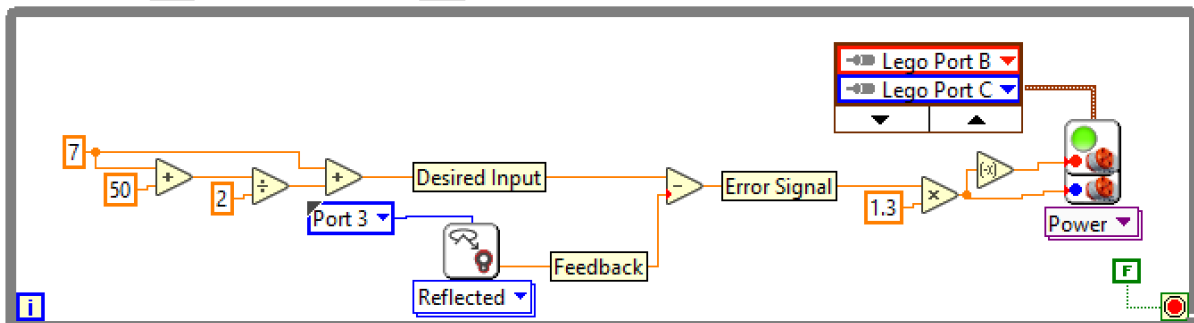
برای این حالت مفهومی به نام فیدبک یا بازخورد را تعریف می‌کنیم. فیدبک یعنی در نظر گرفتن مقدار خروجی سیستم (در اینجا مقدار خوانده شده سنسور) و مقایسه‌ی آن با مقدار ورودی مطلوب (در اینجا مقدار بازتاب روی مرز خط). با اینکار ما عملاً در هر لحظه از وضعیت ربات آگاهی داشته و می‌توانیم آنرا به درستی کنترل کنیم.

بیاید در ابتدا رباتی را در نظر بگیریم که فقط خود را بر روی مرز خط منطبق می‌کند. برای طراحی این ربات لازم است تا ما در هر لحظه مقدار اندازه‌گیری شده‌ی سنسور را بخوانیم و با مقدار مطلوب خود مقایسه کنیم. در حالت کلی به جای مقایسه‌ی این دو مقدار، از تفاضل آنها استفاده می‌شود که به آن سیگنال خطا نیز گفته می‌شود (این سیگنال مشخص می‌کند که سیستم ما در حال حاضر چه مقدار خطا نسبت به حالت مطلوب دارد). پس با تفاضل مقدار مطلوب و مقدار اندازه‌گیری شده، درکی از انحراف ربات از روی مرز خط خواهیم داشت.



شکل ۱۰۰ ایجاد سیگنال خطا با داشتن ورودی مطلوب و خروجی سنسور

حال می‌خواهیم این سیگنال خطا را به گونه‌ای به موتورهای ربات اعمال کنیم تا با چرخش چرخ‌ها، خود را بر روی مرز خط تنظیم کند. هدف ما تنظیم ربات بر روی مرز سمت چپ خط می‌باشد. اگر چرخ چپ به پورت B و چرخ راست به پورت C متصل باشد، با قرار دادن فرمان به موتورها به شکل زیر می‌توان ربات را بر روی مرز سمت چپ خط تنظیم کرد:

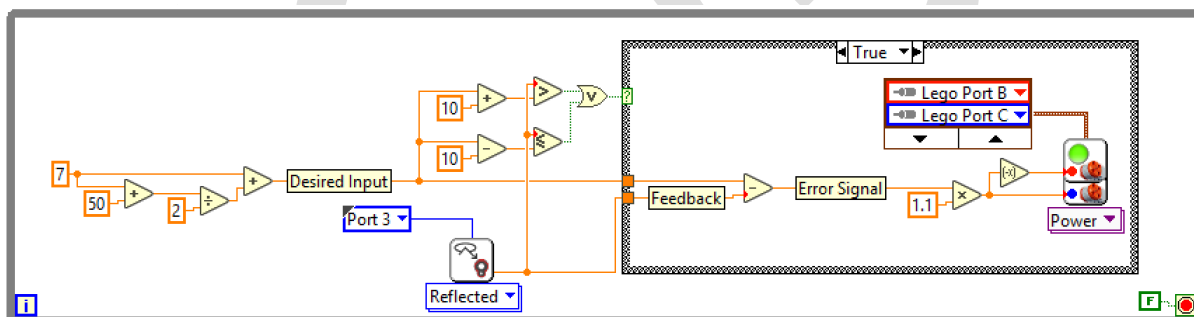


شکل ۱۰۱ چرخش چرخ‌ها متناسب با مقدار سیگنال خطا

در حلقه‌ی بالا ابتدا طبق آنچه که از قبل گفته شده بود، مقدار بازتاب بر روی مرز خط حساب می‌شود، سپس اختلاف آن با مقدار خوانده‌ی سنسور بدست می‌آید که سیگنال خطا را تشکیل می‌دهد. در اینجا برای اینکه سیگنال خطا عددی نه چندان بزرگ برای ورودی به موتورها می‌باشد، از یک ضریب (۱/۳) برای بزرگنمایی

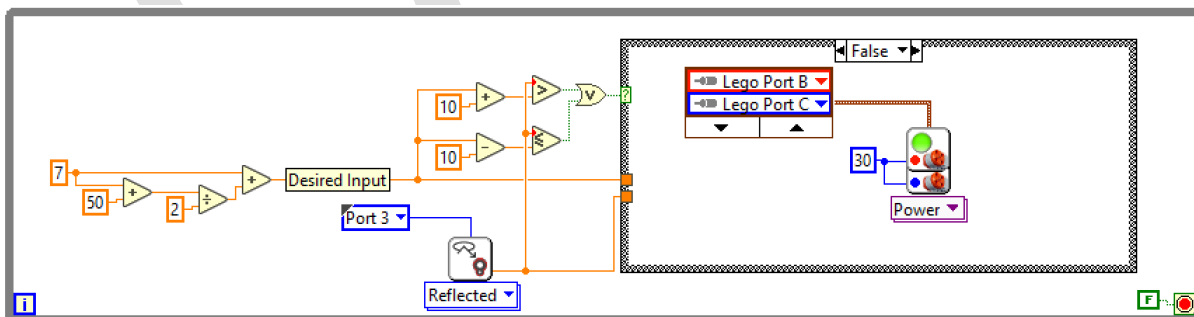
آن استفاده می‌کنیم که در ادبیات کنترل به نام ضریب تناسبی و معمولاً با نماد K_p شناخته می‌شود. در ادامه این مقدار بزرگنمایی شده را به موتور سمت راست و مقدار منفی آنرا به موتور سمت چپ اعمال می‌کنیم. می‌توان ضریب تناسبی را تغییر داد تا بهترین رفتار سیستم را بدست آورد. اما با بزرگ شدن ضریب تناسبی، پایداری سیستم کاهش یافته و احتمال اینکه ربات بتواند خود را بر روی مرز منطبق کند کم می‌شود. از طرفی با کم بودن این ضریب، سرعت تطبیق ربات نیز کم می‌شود. در نتیجه باید حد وسطی را برای این ضریب بدست آورد.

در ادامه می‌خواهیم ربات خط را دنبال کند. برای اینکار، باید یک بازه‌ای حول مقدار مطلوب را به عنوان بازه‌ی مطلوب در نظر گرفت، که تا زمانی که مقدار خوانده شده در این بازه باشد ربات حرکت مستقیم خود را ادامه دهد و زمانی که از این بازه خارج شود، ربات خود را تنظیم کرده و سپس دوباره به درون بازه بازگردد. اگر این بازه‌ی مطلوب را در فاصله‌ی ± 10 از مقدار خوانده شده بگیریم، به این صورت می‌توان در LabVIEW پیاده کرد که اگر بیش از ۱۰ واحد بیشتر از مقدار مطلوب یا کمتر از ۱۰ واحد کمتر از مقدار مطلوب باشد، ربات اقدام به تنظیم خود بر روی خط کند، در غیر این صورت ربات در بازه‌ی مطلوب قرار دارد و می‌توان به حرکت مستقیم خود ادامه دهد. پس در حالتی که ربات در بازه‌ی مطلوب قرار ندارد برای تنظیم ربات مشابه قبل داریم:



شکل ۱۰۲ تنظیم ربات بر روی لبه‌ی خط با کنترل تناسبی

و برای حالتی که ربات در بازه‌ی مطلوب قرار دارد، تنها کافیست ربات به حرکت مستقیم خود ادامه دهد:

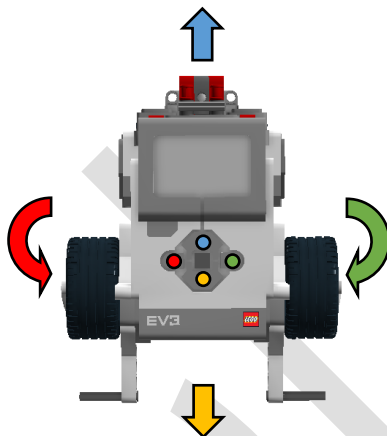


شکل ۱۰۳ حرکت مستقیم ربات در شرایطی که ربات بر روی مرز خط قرار داشته باشد

* برنامه‌ی گفته شده یکی از برنامه‌هاییست که می‌توان برای ربات تعقیب خط، پیاده کرد. واضح است در صورتی که از چند سنسور نور برای شناسایی خط استفاده شود ربات دقیقتر و سریعتر خط را دنبال می‌کند.

تمرین فصل سوم

۱. برای ربات پایه، برنامه‌ای طراحی کنید تا با استفاده از دکمه‌های روی بریک حرکت نماید به این صورت که با فشردن کلیدهای بالا، پایین، چپ و راست به ترتیب به سمت جلو، عقب حرکت کرده و چرخش به چپ و راست را انجام دهد.



تمرین ۱ حرکت ربات با کلیدهای موجود بر روی بریک

۲. بر روی یک سطح سفید رنگ، با استفاده از نوار چسب مشکی (قابل تشخیص با سنسور نور) مسیر پیچیده‌ای را طراحی کنید. حال با استفاده از مطالب گفته شده ربات تعقیب خطی را طراحی کنید تا مسیر را به خوبی دنبال کند. با سعی و خطا بهترین ضریب کنترلر را پیدا کنید تا ربات بهترین تعقیب خط را انجام دهد.



تمرین ۲ تعقیب خط با تنظیم ضریب تناسبی

۳. ربات ترکیبی تعقیب خط و تعقیب دیوار را بسازید (هر دو سنسور نور و فراصوت را بر روی ربات خود تعبیه کنید). مسیر ترکیبی شامل خطوط مشکی بر روی سطح سفید و همچنین دیواره طراحی کنید تا ربات بتواند با استفاده از تعقیب خط و دیوار، مسیر را طی کند. حال برنامه‌ای را بنویسید تا ربات به خوبی قادر به پیوندن مسیر ترکیبی باشد (ربات باید در هر لحظه بررسی کند که آیا خط وجود دارد یا دیوار، سپس با استفاده از الگوریتم نوشته شده، مسیر را دنبال نماید).



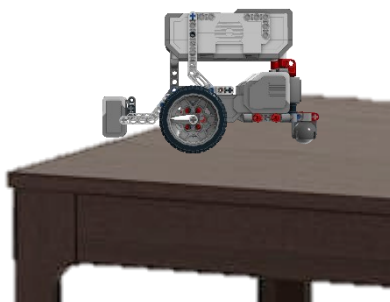
تمرین ۳ مسیریابی ربات با کمک تعقیب خط و دیوار

۴. رباتی را ساخته و برنامه‌نویسی کنید که با اندازه‌گیری فاصله از مانع، صدای بوقی را متناسب با آن فاصله پخش نماید. با تغییر فاصله‌ی مانع، فرکانس بوق تولید شده متناسب با آن تغییر نماید.



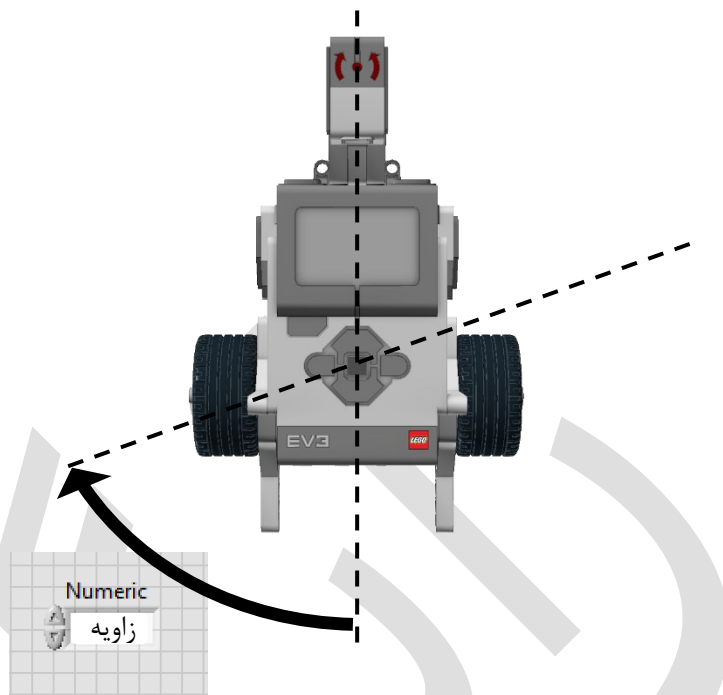
تمرین ۴ تولید صدای بوق متناسب با فاصله‌ی مانع

۵. ربات شناسایی پرتگاهی طراحی کنید که بر روی یک میز بدون دیواره قرار گرفته و در صورت رسیدن به لبه‌ی میز، پرتگاه را تشخیص داده و مسیر خود را به گونه‌ای تغییر دهد که سقوط ننماید.



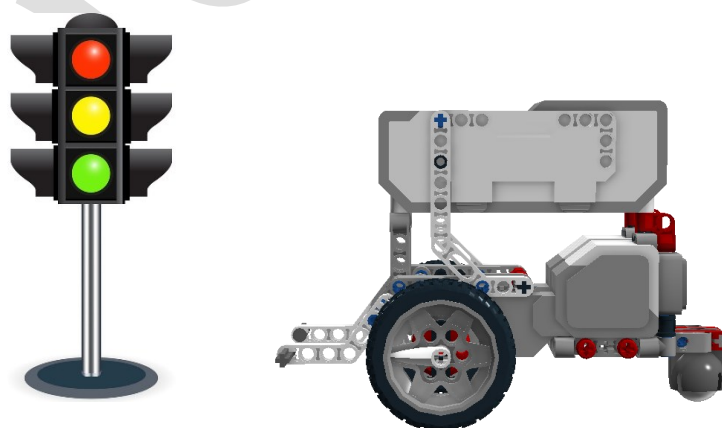
تمرین ۵ تشخیص پرتگاه و جلوگیری از سقوط ربات

۶. رباتی را طراحی کنید که با تعیین زاویه‌ی قرارگیری ربات از Front Panel، چرخش نموده و در آن جهت قرار گیرد.



تمرین ۶ چرخش ربات با مقدار فرمان داده شده

۷. با استفاده از سنسور رنگ، رباتی را طراحی کنید که با نمایش رنگ سبز به آن حرکت نموده و با نمایش رنگ قرمز به آن، متوقف شود.



تمرین ۷ حرکت ربات با نور سبز و توقف با نور قرمز

فصل چهارم

«برنامه نویسی پیشرفته MINDSTORMS»

مقدمه

در این فصل از این کتاب قصد داریم تا با ساخت و برنامه‌نویسی ربات‌های پیشرفته‌تر، مهارت‌های برنامه‌نویسی خود را ارتقا داده و مسائل و چالش‌های پیش رو را به اهداف واقعی نزدیک‌تر نماییم. در هر یک از ربات‌هایی که در ادامه آورده می‌شوند، ابتدا هدف و وظیفه‌ی ربات ذکر شده و سپس روند ساخت ربات معرفی و شرح داده می‌شود و در نهایت به برنامه‌نویسی آن پرداخته خواهد شد.

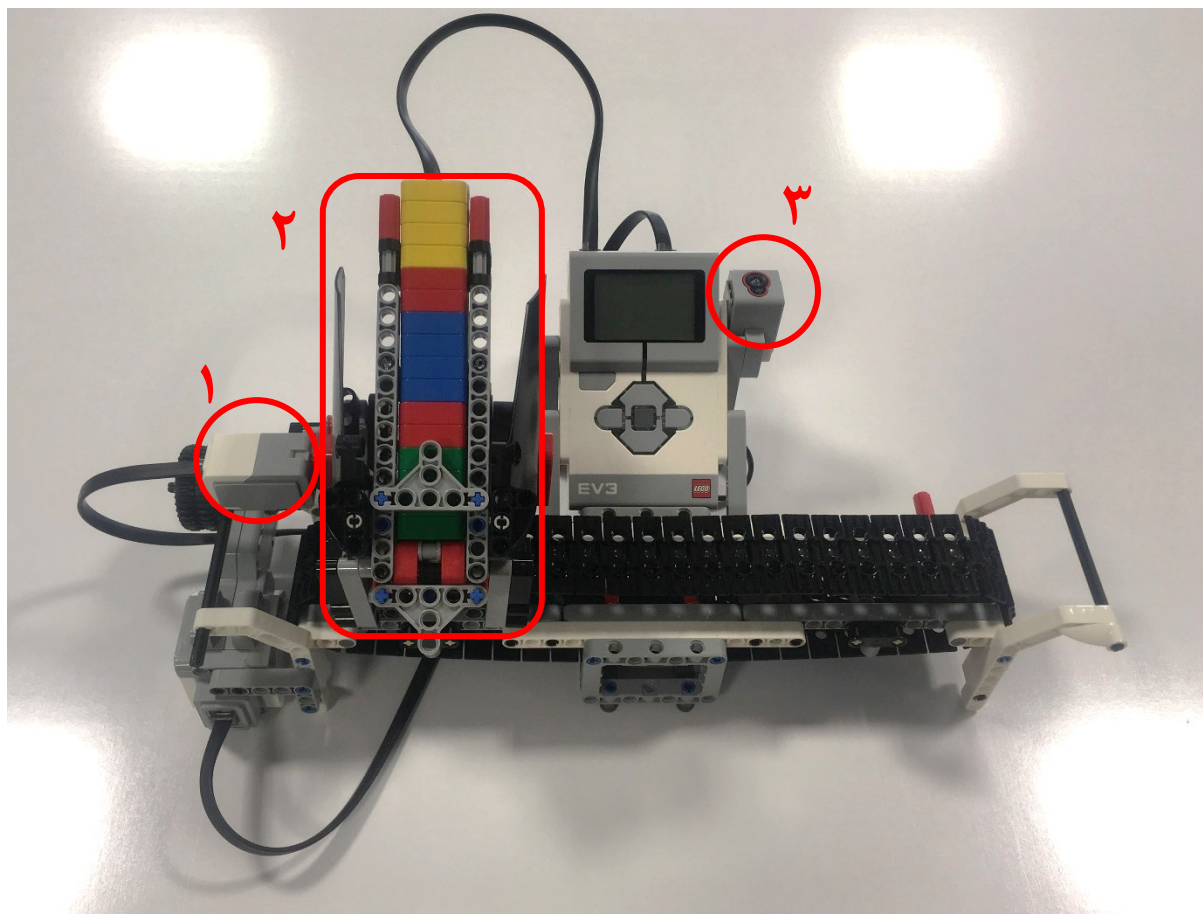
برای برنامه‌نویسی ربات‌ها می‌توان روش‌های مختلفی را به کار برد. در نتیجه برنامه‌ی گفته شده در این کتاب تنها برنامه‌ای نیست که می‌توان برای این ربات‌ها طراحی نمود. پیشنهاد می‌شود برای یادگیری بهتر و بیشتر، بعد از ساخت ساختار ربات، با دقت و حوصله روند برنامه‌نویسی را تعیین نموده و شروع به طراحی برنامه آن نمایید، در نهایت برای بررسی و صحت‌سنجی برنامه خود به کتاب مراجعه نموده تا از تمامی نکات موجود در این بخش بهره ببرید.

ربات تفکیک رنگ (Color Sorter)

در این قسمت هدف ما ساخت رباتی است که بلوک‌های رنگی را شناسایی کرده و هر بلوک را با توجه به رنگ آن، در دسته‌بندی جدا قرار دهد. ابتدا به ساخت این ربات با توجه به دستورالعمل موجود در پیوست انتهای کتاب خواهیم پرداخت.

بعد از اتمام مراحل ساخت ربات، نوبت برنامه‌نویسی آن می‌باشد.

برای آشنایی بیشتر با این ربات، به معرفی قطعات و اجزای این ربات می‌پردازیم:

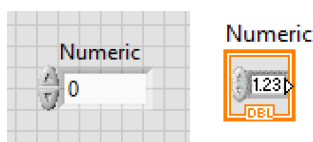


شکل ۱ نمای کلی و بخش‌های اصلی ربات تفکیک رنگ

۱. سنسور تماس: برای بازگشت ارابه به موقعیت اولیه استفاده می‌گردد.
۲. ارابه: برای حمل بلوک‌ها و تخلیه بلوک‌ها استفاده می‌گردد که عمل تخلیه توسط موتور متصل به پورت A انجام می‌گردد. جابجایی ارابه بر روی تسمه توسط موتور متصل به پورت D انجام می‌گیرد.
۳. سنسور نور/رنگ: برای شناسایی رنگ بلوک‌ها استفاده می‌گردد.

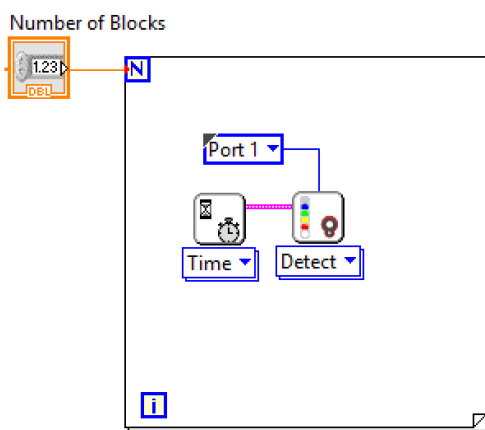
لازم است تا مراحل عملیات ربات را به چند بخش تقسیم کنیم: شناسایی، تصمیم‌گیری، حرکت. در ابتدا لازم است تا بلوک‌های مورد نظر توسط ربات شناسایی شوند. در نتیجه باید قبل از قرار دادن هر بلوک رنگی در درون مخزن ربات، ربات رنگ آنرا تشخیص داده و ذخیره نماید. برای تشخیص رنگ از سنسور نور/رنگ استفاده می‌شود. پس طرز کار ربات تا به اینجا به این صورت است که هر بلوک ابتدا در روبروی سنسور رنگ

ربات نگه داشته شده تا شناسایی رنگ انجام شود، سپس این بلوک به درون مخزن ربات قرار داده شود. بدین ترتیب بلوک‌های رنگی به ترتیب شناسایی شده و به همان ترتیب درون مخزن ربات قرار داده می‌شود. پس برای شروع برنامه‌نویسی این ربات، برنامه‌نویسی مرحله شناسایی را انجام می‌دهیم. برای این منظور از آرایه‌ها برای ذخیره‌ی رنگ هر بلوک استفاده می‌نماییم. از آنجایی که تعداد بلوک‌ها ممکن است تغییر کند، تعداد بلوک‌ها را نیز در پنجره Front Panel از کاربر دریافت می‌نماییم، برای این منظور از Numeric Control استفاده می‌نماییم:



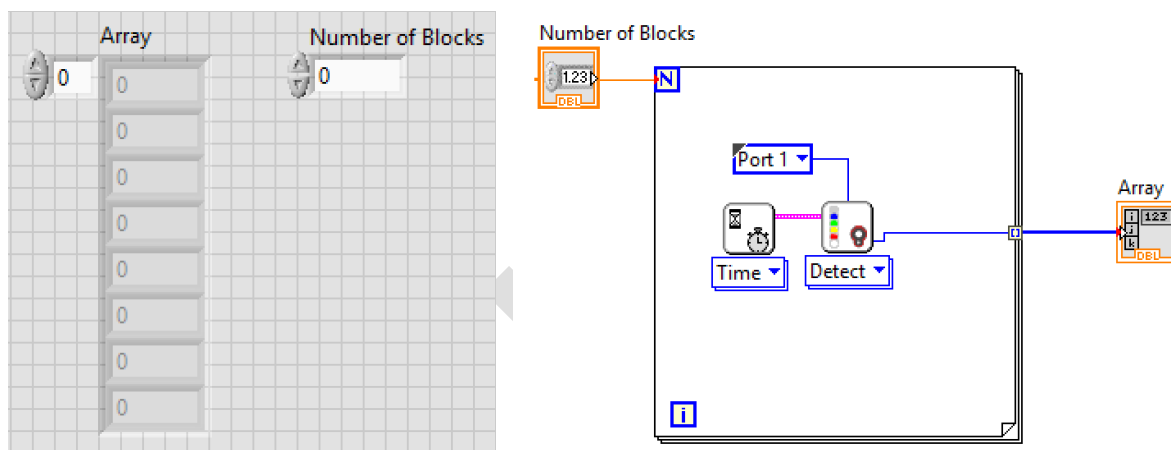
شکل ۲ ایجاد ورودی عددی

مقدار وارد شده در این ورودی عددی بیانگر تعداد بلوک‌ها خواهد بود. در مرحله اول قرار است تا بلوک‌ها را شناسایی نموده و رنگ آنها را به ترتیب شناسایی شده، در متغیری ذخیره نماییم. از آنجایی که قرار است روند شناسایی و ذخیره به تعداد بلوک‌ها تکرار شود پس باید از یک حلقه for برای این کار استفاده نمود. با توجه به مطالب گفته شده در مبحث سنسور رنگ/نور، حلقه for را به شکل زیر تشکیل می‌دهیم:



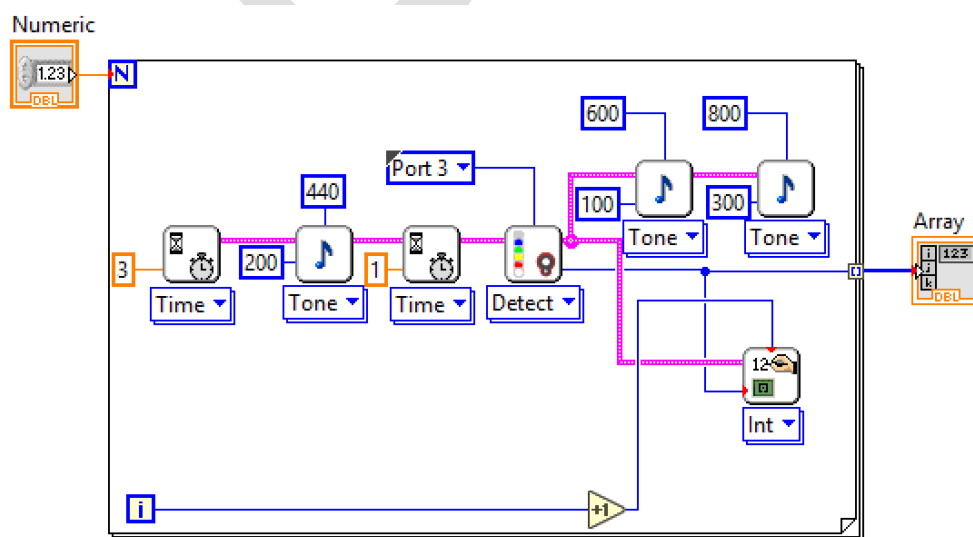
شکل ۳ خواندن رنگ بلوک‌ها به تعداد مشخص شده

در قدم بعد باید مقادیر خوانده شده توسط سنسور رنگ/نور در متغیری ذخیره شوند تا بعدا بتوان به این مقادیر دسترسی داشت. در نتیجه با استفاده از یک آرایه در پنجره Front Panel و قرار دادن شاخص عددی (Numeric Indicator) در آن، می‌توان خروجی سنسور را در این آرایه ذخیره نمود:



شکل ۴ ذخیره‌ی خروجی سنسور رنگ در یک آرایه

حال مرحله شناسایی کامل شده است. اما با توجه به اینکه روند شناسایی بدون هیچ وقفه‌ای انجام می‌شود برنامه‌ی نوشته شده کاربردی نمی‌باشد زیرا برنامه در یک لحظه به تعداد بلوک‌ها شناسایی را انجام داده و به اتمام می‌رسد. برای کاربردی بودن برنامه باید با استفاده از صفحه‌نمایش و بلندگو، نشانه و علائمی را به کاربر نشان دهیم تا کاربر و برنامه به ترتیب روند شناسایی را انجام دهند. با کمی حوصله و توجه به مطالب گفته شده، برنامه را می‌توان به شکل زیر ارتقا داد:



شکل ۵ تکمیل برنامه‌ی خواندن و ذخیره کردن رنگ بلوک‌ها

در این حالت ابتدا بریک به مدت ۳ ثانیه توقف کرده و بعد از آن صدای بوقی را به مدت ۲۰۰ میلی‌ثانیه پخش کرده و دوباره به مدت ۱ ثانیه توقف کرده تا کاربر بلوک را روبروی سنسور رنگ قرار دهد. بعد از شناسایی رنگ بلوک، بریک صدایی را به معنی اتمام شناسایی پخش می‌نماید. همزمان با انجام شناسایی، کد رنگ بلوک

شناسایی شده بر روی صفحه نمایش بزرگ نشان داده می‌شود. این عملیات تا جایی انجام می‌شود تا به تعداد بلوک‌ها شناسایی انجام شده و کد رنگ‌ها در یک آرایه ذخیره شود.

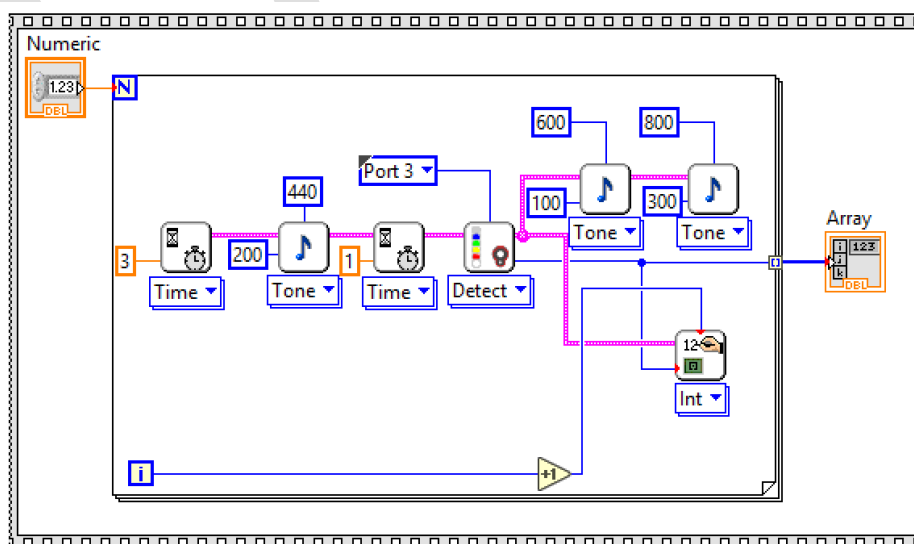
در مرحله‌ی بعد باید بلوک‌ها را به ترتیب شناسایی شده در مخازن مناسب قرار دهیم. برای این کار باید دو مورد را مشخص نماییم. مورد اول مقدار حرکت ارابه با کمک حرکت تسمه می‌باشد که با توجه به اینکه طول مسیر ارابه محدود و تعداد رنگ بلوک‌ها برابر ۴ می‌باشد، می‌توان مقدار حرکت ارابه را مشخص نمود. مورد دوم مقدار چرخش موتور A برای تخلیه یک بلوک از داخل مخزن ارابه می‌باشد.

برای مشخص کردن طول مسیر حرکت ارابه، می‌توان با یک برنامه ساده، موتور D را چند دور کامل چرخاند تا تعیین کنیم طول مسیر ارابه در چند دور حرکت موتور طی می‌شود. اگر از این روش استفاده کنیم درمی‌یابیم که با چرخش ۵۴۰ درجه (۱.۵ دور)، ارابه تمام مسیر تسمه را طی می‌کند. در نتیجه باید این مسیر را به ۴ قسمت تقسیم کنیم تا در هنگام جداسازی بلوک‌های رنگی، هر کدام در موقعیتی مشخص تخلیه شود.

برای تعیین مقدار چرخش موتور A برای تخلیه هر بلوک، اگر به مکانیزم این ربات دقت نماییم متوجه می‌شویم با هر بار چرخش کامل این موتور، یک بلوک از مخزن ارابه تخلیه می‌شود. پس برای هر بار تخلیه کردن بلوک، لازم است تا موتور A را یک دور کامل (۳۶۰ درجه) حرکت دهیم.

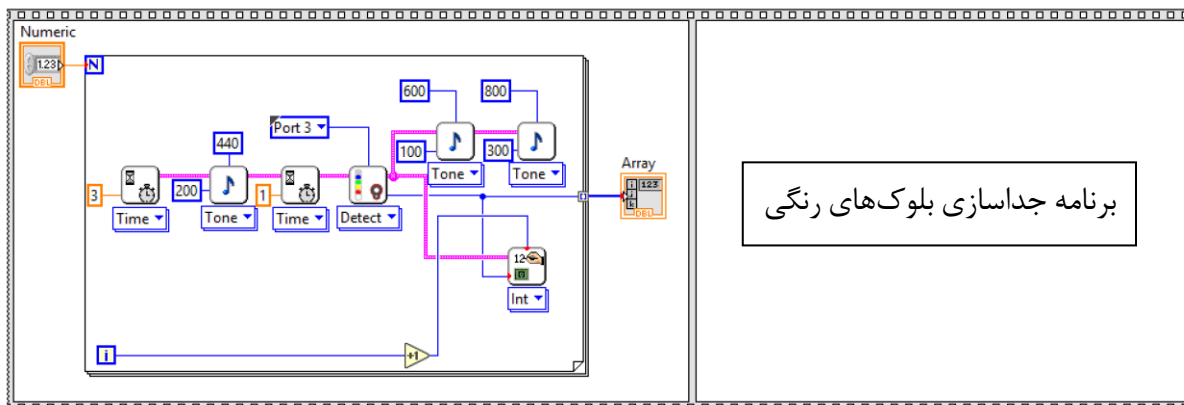
در مرحله‌ی جداسازی، لازم است که ابتدا رنگ ذخیره شده در آرایه‌ی ساخته شده، خوانده شود و سپس با توجه به کد رنگ، ارابه را در فاصله‌ی مناسبی از سنسور تماس برده و بلوک رنگی تخلیه شود و دوباره به محل اولیه بازگردد تا برای تخلیه بلوک رنگی بعدی آماده شود.

نکته‌ی مهمی که در اینجا وجود دارد این است که ابتدا باید برنامه، روند شناسایی و ذخیره رنگ را انجام داده و سپس به سراغ روند جداسازی برود. برای این منظور همانطور که گفته شد از ساختار Flat Sequence استفاده می‌نماییم. در این صورت ابتدا روند شناسایی و ذخیره رنگ را در فریم اول قرار داده و سپس فرآیند جداسازی را در فریم بعد قرار می‌دهیم تا این دو روند به ترتیب و بدون تداخل با یکدیگر اجرا شوند. پس برنامه‌ای که تا به حال برای شناسایی و ذخیره رنگ طراحی شده است را درون یک فریم Flat Sequence قرار می‌دهیم:



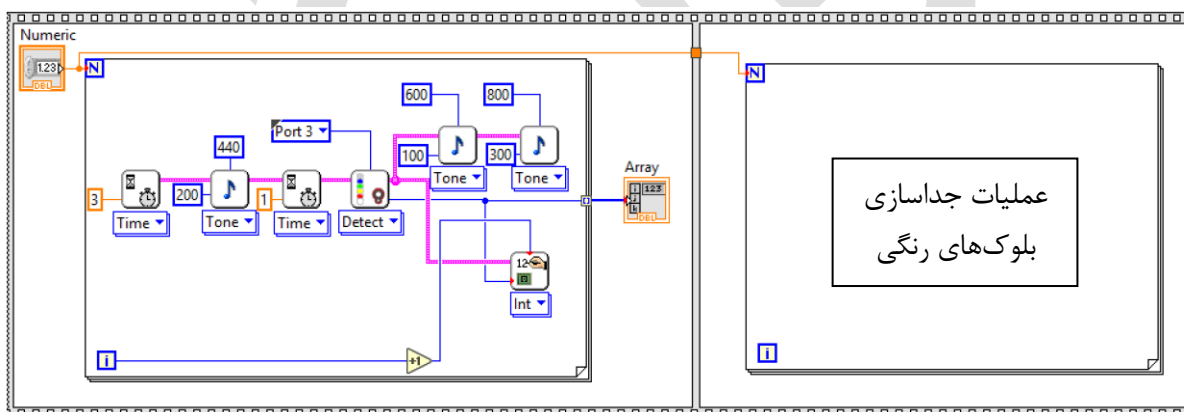
شکل ۶ قرار دادن برنامه در فریم اول Case Structure

برای ساخت روند جداسازی باید طبق مطالب گفته شده یک فریم به ساختار Flat Sequence اضافه شود و در فریم جدید این روند طراحی گردد:



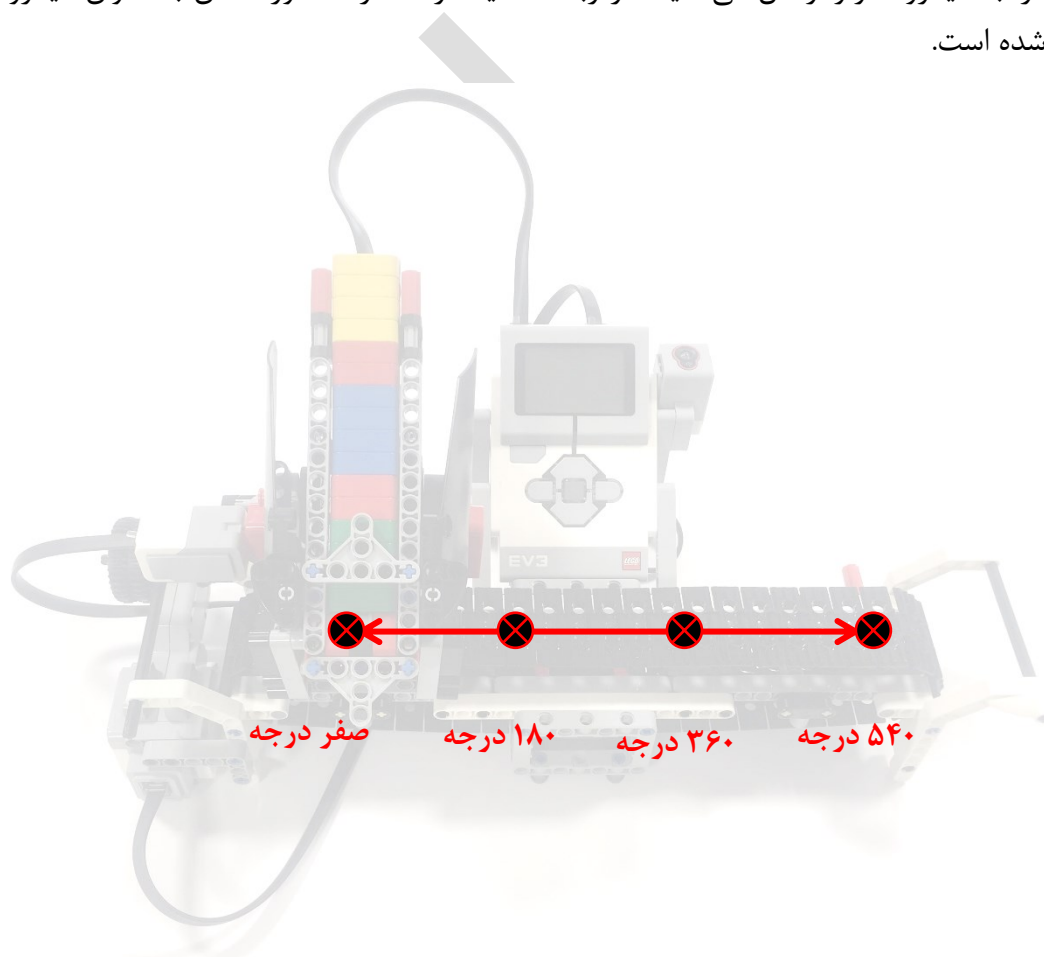
شکل ۷ ایجاد فریم جدید برای طراحی برنامه جداسازی بلوک‌ها

همانطور که می‌دانیم فرآیند جداسازی یک فرآیند تکرار شونده است که باید به تعداد بلوک‌های خوانده شده، این عمل تکرار شود. در نتیجه بهترین راه استفاده از حلقه For می‌باشد تا به تعداد بلوک‌های خوانده شده، عملیات جداسازی تکرار شود:



شکل ۸ تکرار عملیات جداسازی در فریم دوم

در اکثر سیستم‌های اتوماتیک و مکانیزه، اغلب از میکروسوییچ‌ها برای برگرداندن ماشین یا ربات به حالت اولیه (که حالت مرجع و شناخته شده‌ای می‌باشد) استفاده می‌شود. هرچند که اغلب ربات‌ها یا سیستم‌ها از موتورهای استپر (موتورهایی که زاویه‌ی چرخش مشخصی دارند) و موتورهای سروو (موتورهایی که در آنها انکودر یا پتانسیومتر برای اندازه‌گیری زاویه و سرعت وجود دارد) استفاده می‌شود، اما به علت اینکه سیستم‌های فیزیکی عمدتاً دارای خطاهای کوچکی می‌باشند و یا ممکن است در هنگام کار دستگاه، عملیات مورد نظر آنطور که انتظار می‌رود انجام نشود، از میکروسوییچ‌ها استفاده می‌شود. میکروسوییچ یک کلید ساده می‌باشد که بر روی چهارچوب و بخش‌های ثابت دستگاه قرار می‌گیرد و در هنگام تماس با بخش‌های متحرک دستگاه، سیگنالی را به میکروکنترلر ارسال می‌نماید. در ربات تفکیک رنگ، از سنسور تماس به عنوان میکروسوییچ استفاده شده است.

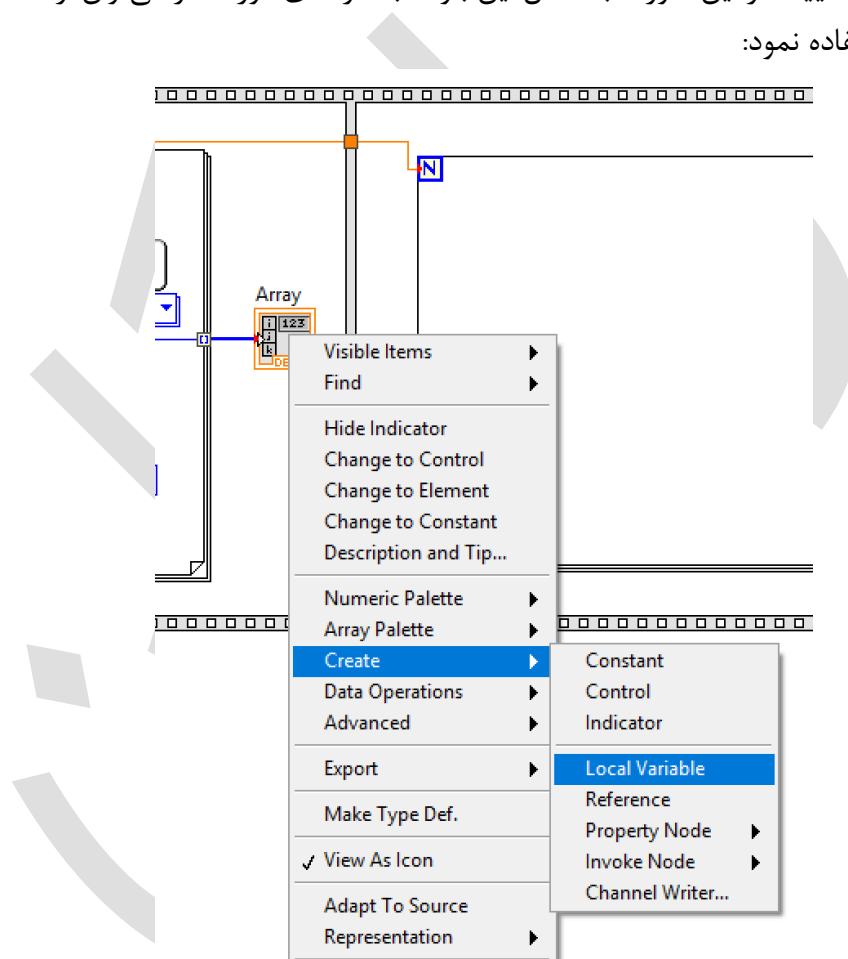


شکل ۹ مقدار چرخش لازم برای تفکیک بلوک‌ها در ۴ مخزن متفاوت

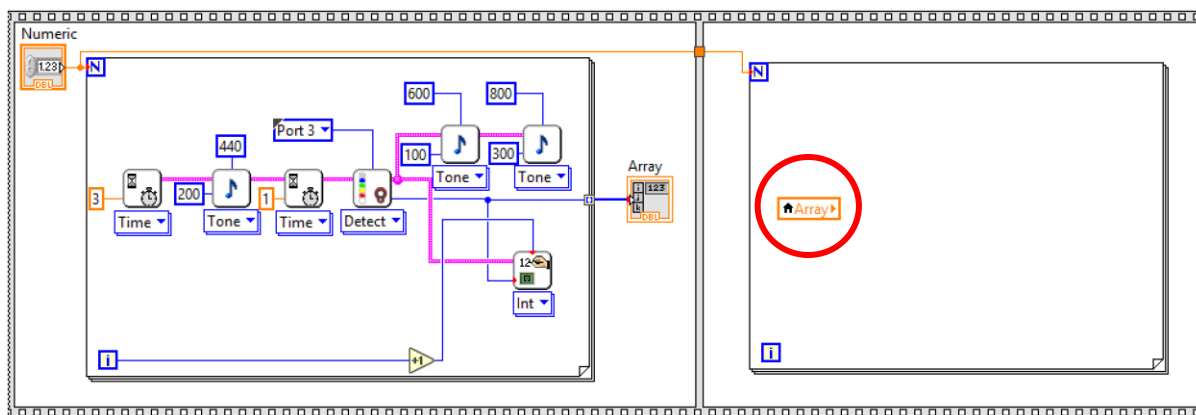
برای اینکه بعد از هر بار حرکت ارابه، بتوانیم حرکت‌های بعدی ارابه را به درستی و دقت انجام دهیم، لازم است تا آنرا به موقعیت اولیه حرکت دهیم، به این صورت که ارابه را آنقدر به سمت سنسور تماس حرکت دهیم تا سنسور تماس فعال شود. با انجام اینکار خطاهای کوچکی که در اندازه‌گیری و یا حرکت موتور وجود دارد حذف شده و ربات ما کارکرد بی نقصی را از خود نشان می‌دهد.

پس روند جداسازی به این صورت است که ابتدا کد رنگ خوانده شده برای بلوک مورد نظر از آرایه ذخیره شده، خوانده می‌شود، سپس با توجه به رنگ بلوک، ارابه را به مقدار مناسب حرکت داده و بعد از رسیدن به

محل مناسب، بلوک از داخل مخزن ارایه، تخلیه می‌شود. در نهایت ارایه را به سمت موقعیت اولیه خود حرکت می‌دهیم تا هنگامی که ارایه به سنسور تماس برخورد کرده و سپس آنرا متوقف می‌نماییم. برای دسترسی به داده‌های ذخیره شده در یک متغیر یا آرایه، می‌توان از بلوک متغیر محلی (Local Variable) استفاده نمود. برای اینکه بلوک متغیر محلی مربوط به یک متغیر یا آرایه ایجاد شود، تنها لازم است بر روی متغیر یا آرایه مورد نظر کلیک راست کرده و گزینه Create→Local Variable را انتخاب نماییم و سپس بلوک ایجاد شده را در فضای مناسب قرار دهیم. توجه نمایید که بلوک ایجاد شده به حالت ورودی قرار داده می‌شود، برای اینکه این بلوک به حالت خروجی دربیاید با راست کلیک بر روی آن و انتخاب گزینه Change To Read را انتخاب نمایید. در این صورت با اتصال این بلوک به گره‌های مورد نظر می‌توان از داده‌های داخل آن متغیر یا آرایه استفاده نمود:

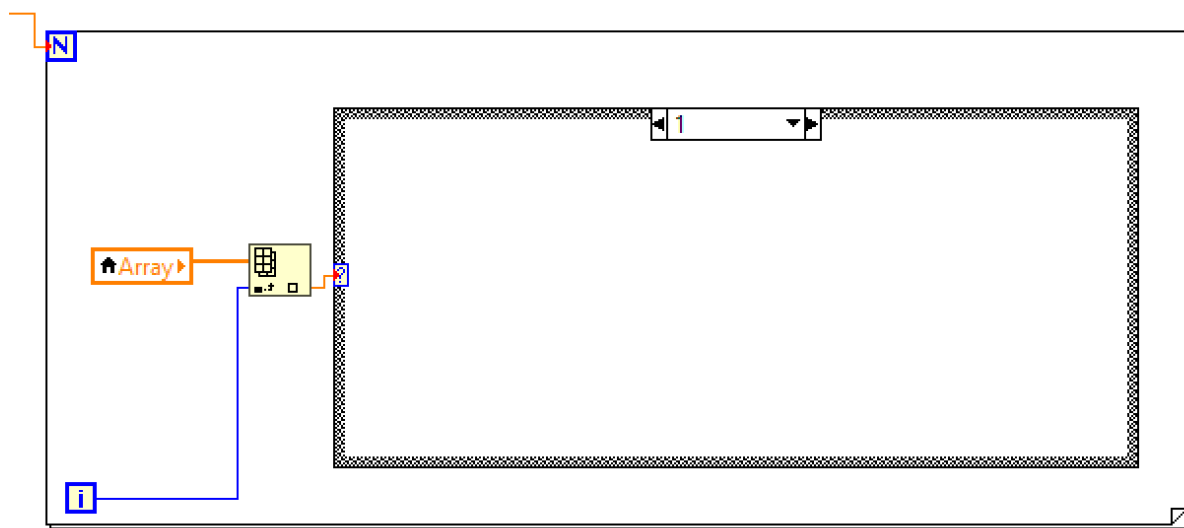


شکل ۱۰ ایجاد یک متغیر محلی



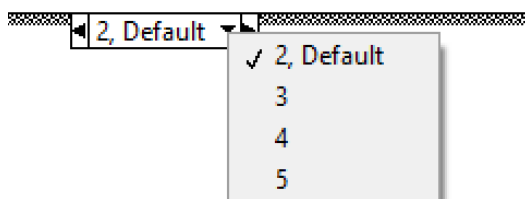
شکل ۱۱ ایجاد ترمینال متغیر محلی

حال که به مقادیر داخل آرایه دسترسی داریم می‌توان با خواندن به ترتیب این مقادیر، در خصوص تفکیک بلوک‌های رنگی تصمیم بگیریم. همانطور که در قسمت‌های قبل گفته شد با استفاده از بلوک Index Array می‌توان با مشخص کردن اندیس هر درایه، مقدار آنرا استخراج کرد. پس برای اینکه به مقادیر مختلف ذخیره شده در آرایه دسترسی داشته باشیم از این بلوک به شکل زیر استفاده می‌نماییم:



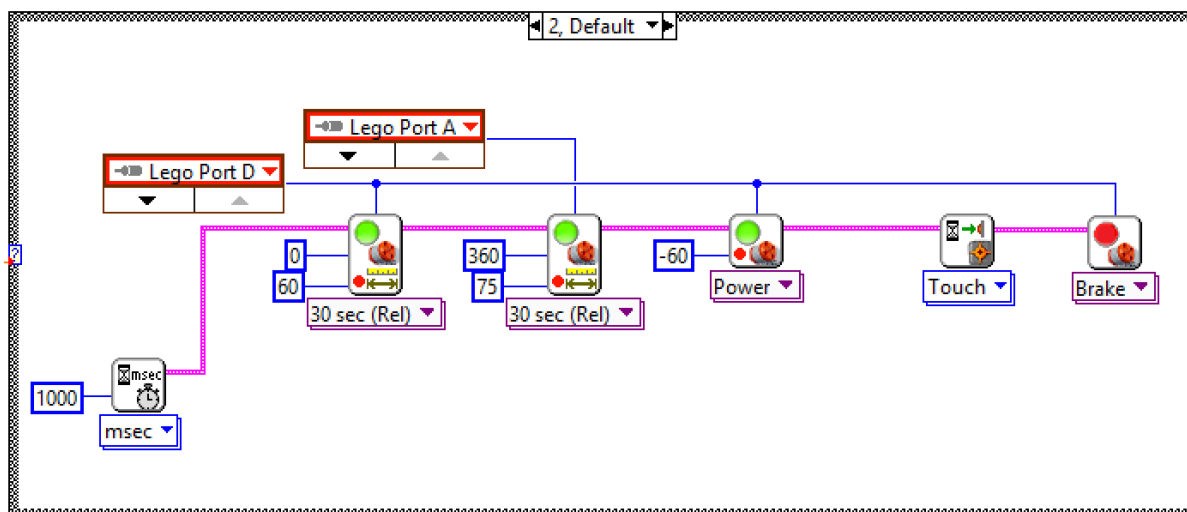
شکل ۱۲ خواندن کدهای رنگ ذخیره شده در آرایه

از آنجایی که هر کد نشانگر رنگ خاصی می‌باشد، در اینجا چهار رنگ آبی، سبز، زرد و قرمز داریم که به ترتیب دارای کدهای ۲، ۳، ۴ و ۵ می‌باشند. پس موردهای موجود در Case Structure را متناسب با این اعداد به شکل زیر قرار می‌دهیم:



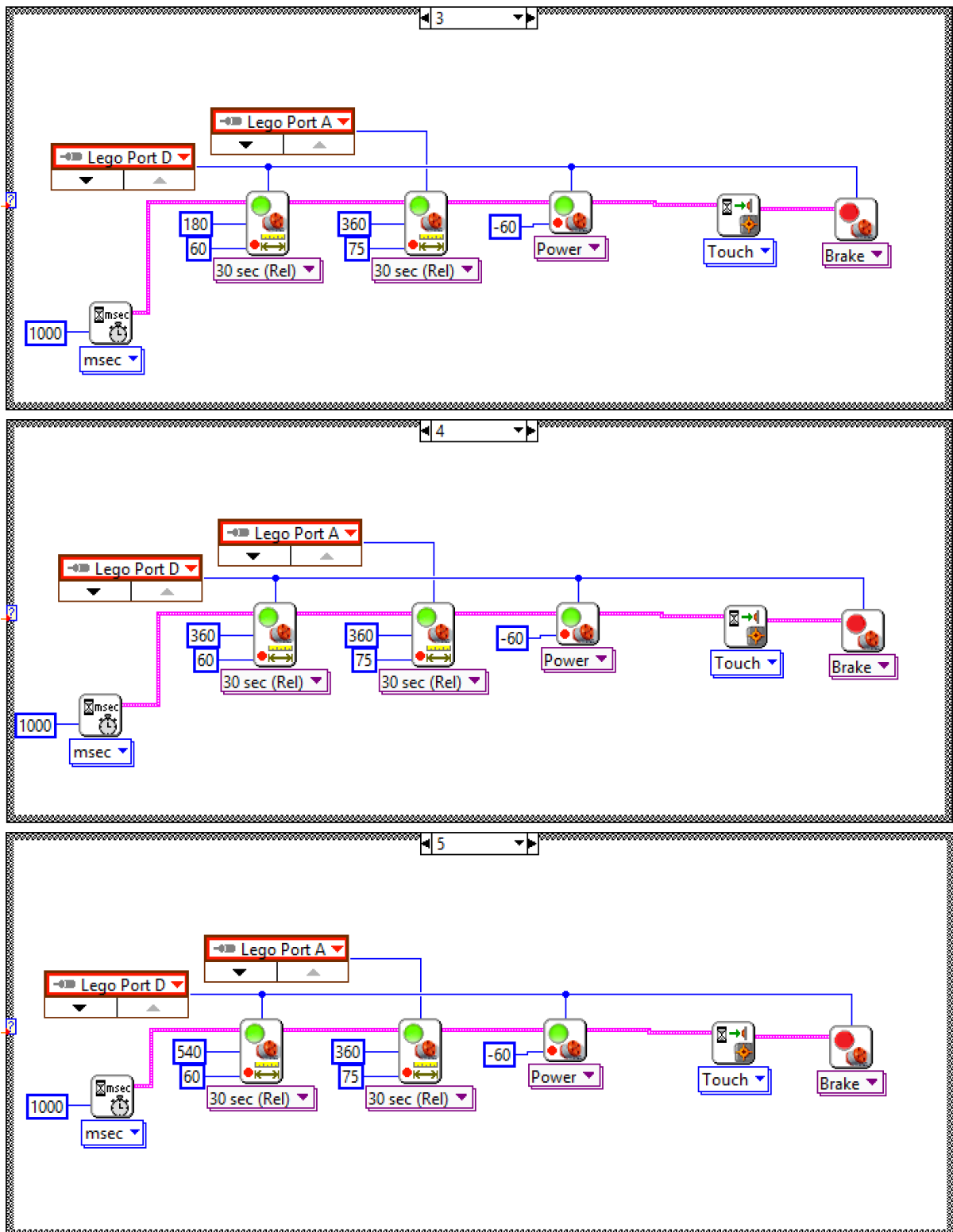
شکل ۱۳ ایجاد موردهای مختلف برای تشخیص و تفکیک بلوک‌های رنگی

در داخل هر مورد باید به ترتیب عملیات حرکت ارابه به نقطه مورد نظر، تخلیه بلوک و بازگشت ارابه به محل اولیه ایجاد شود. تنها تفاوتی که این عملیات برای رنگ‌های مختلف (موردهای مختلف) دارد مقدار چرخش موتور D برای حرکت ارابه می‌باشد که در قسمت قبل میزان چرخش آن توضیح داده شد. در نتیجه برنامه‌ی موجود در قسمت Case Structure به شکل زیر طراحی می‌گردد:



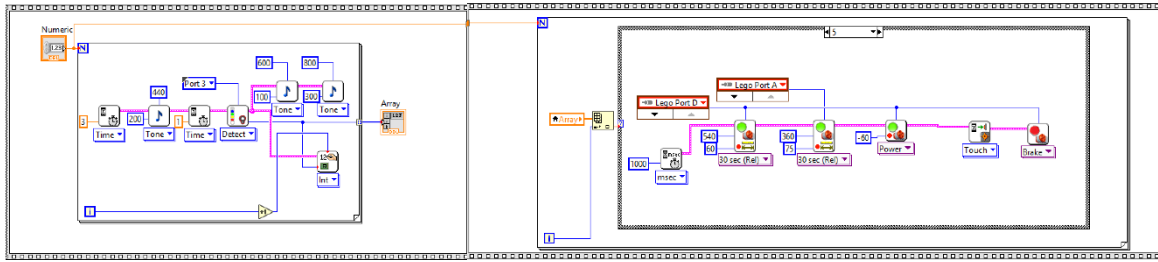
شکل ۱۴ طراحی برنامه برای حرکت ارابه به نقطه مورد نظر

برای عملکرد بهتر این ربات، مقدار تاخیر ۱ ثانیه‌ای در ابتدای هر عملیات تفکیک را قرار داده‌ایم. برای سایر موردها تنها مقدار چرخش موتور D تغییر می‌نماید. برای سایر موردها برنامه به شکل زیر تغییر می‌کند:



شکل ۱۵ تغییر مقدار چرخش موتور D متناسب با رنگ خوانده شده در هر مورد

حال طراحی برنامه ربات به اتمام رسیده و ربات ابتدا بلوک‌های رنگی را تشخیص داده و سپس آنها را در جای مناسب خود تفکیک می‌نماید. برنامه‌ی نهایی به شکل زیر می‌باشد:



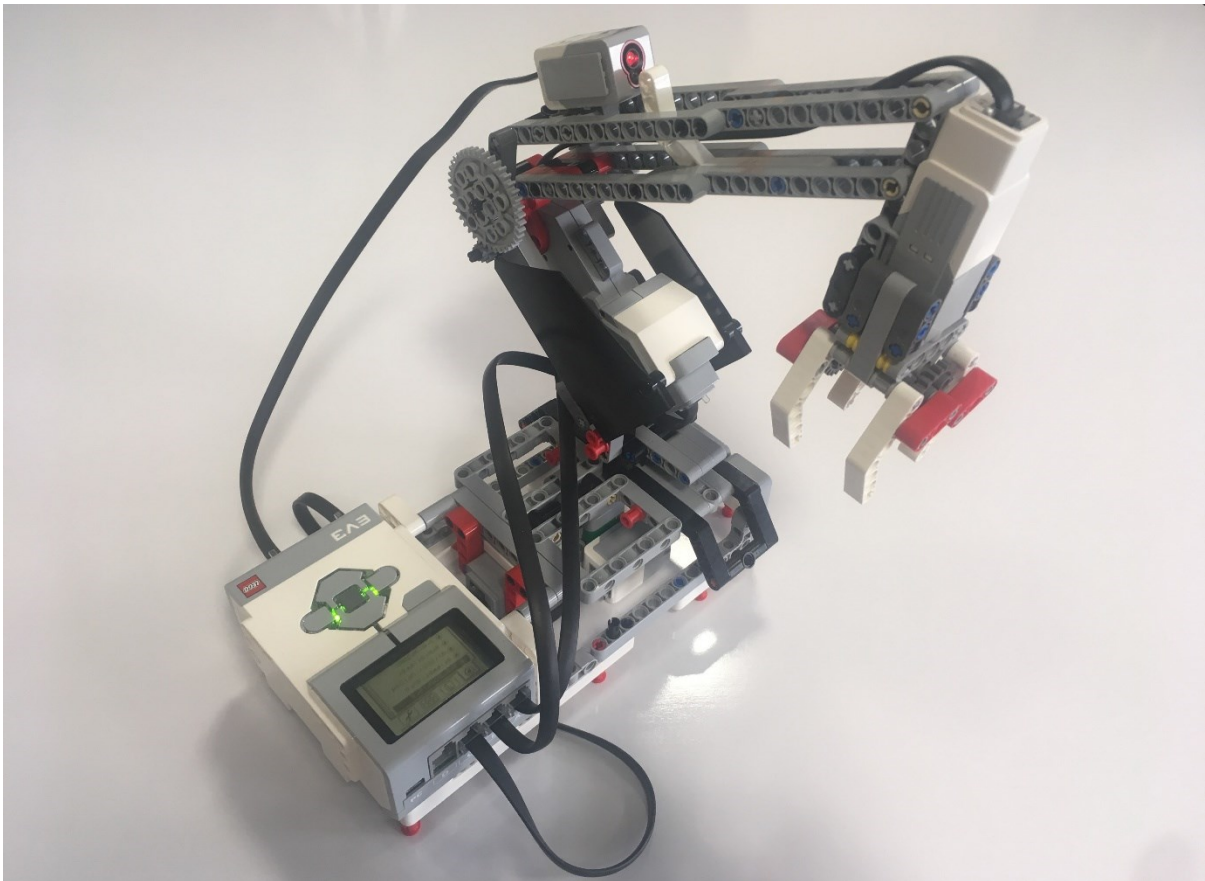
شکل ۱۶ برنامه‌ی کامل ربات تفکیک رنگ

دقت نمایید که هر یک از مراحل گفته شده در بالا را می‌توان به روش‌های مختلفی انجام داد و لزومی ندارد تا حتماً به شکل گفته شده انجام شود و از طرفی با تغییر در برنامه و حتی ساختار ربات نیز می‌توان عملکرد آنرا بهبود بخشید.

تمرین ربات تفکیک رنگ

۱. برای تمرین بیشتر برنامه‌نویسی ربات را به گونه‌ای تغییر دهید تا تعداد بلوک‌ها از پیش معلوم نبوده و با زدن یک دکمه عملیات شناسایی متوقف شده و عملیات تفکیک شروع گردد.
۲. برنامه‌ی ربات را به گونه‌ای طراحی کنید تا در هنگام جداسازی بلوک‌ها، هر دفعه به موقعیت اولیه باز نگردد و به موقعیت بلوک بعدی برود. آیا همچنان ربات دقت لازم برای انجام عملیات جداسازی را دارد؟
۳. ساختار ربات را به گونه‌ای تغییر دهید تا ابتدا تمامی بلوک‌ها را در مخزن قرار داده و سپس پیش از جداسازی بلوک‌ها، رنگ آخرین بلوک را بخواند و آنرا در موقعیت مناسب تخلیه کند.

بازوی رباتیک Robot Arm



شکل ۱۷ بازوی رباتیک

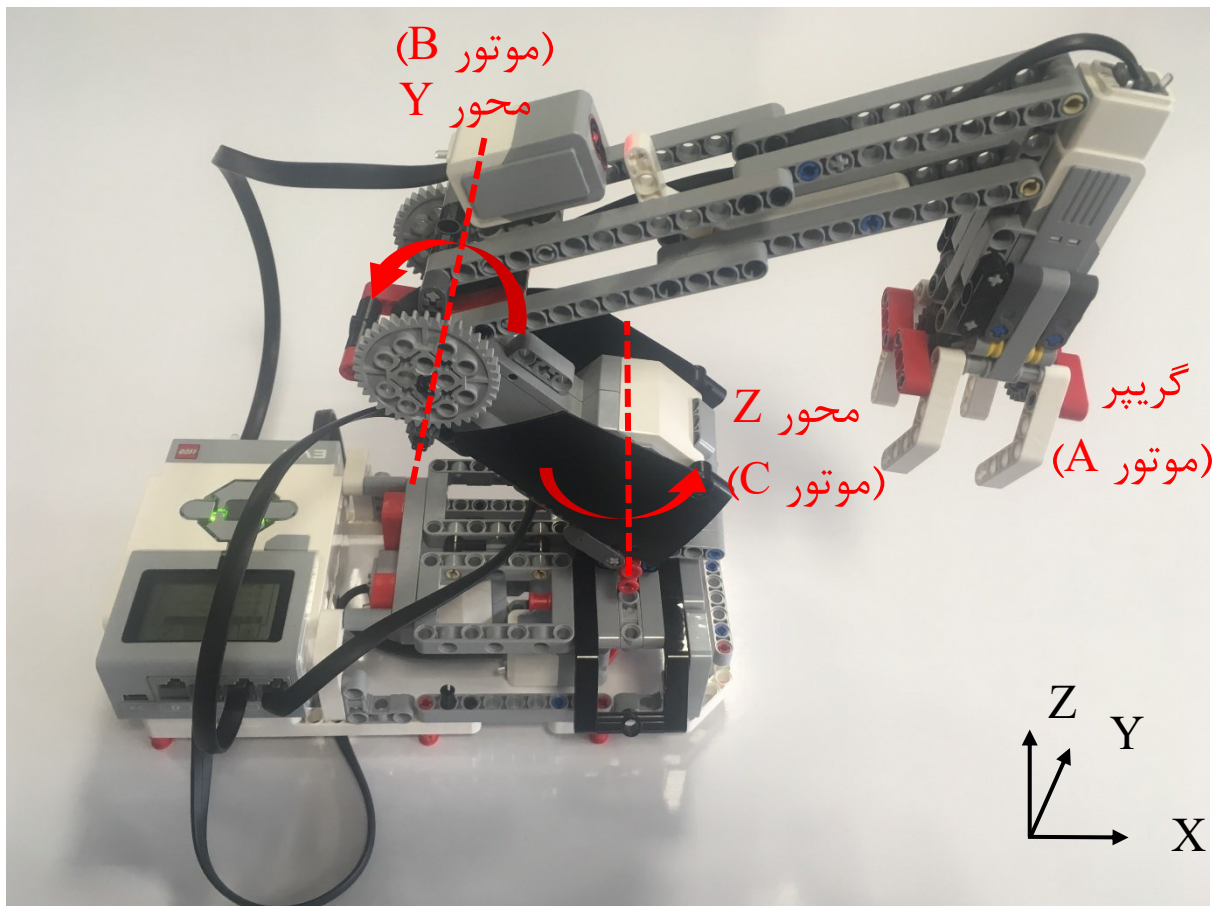
یکی از معروفترینها و پرکاربردترین ماشینها در صنایع و دنیای رباتیک، بازوی رباتیک می باشد. بازوهای رباتیک از چندین عضو تشکیل شده اند که این اعضا با مفاصل لولایی یا کشویی نسبت به یکدیگر و با استفاده از عملگرهای گوناگون، حرکت کرده و کنترل می شوند. همچنین برای کنترل حرکت بازوها و عدم برخورد اعضای آن با یکدیگر و آسیب رساندن، از سنسورهای گوناگونی استفاده می گردد.

بازوهای رباتیک می توانند از دو نوع مکانیزم سری یا موازی تشکیل شده باشند. مکانیزم سری مکانیزمی است که نقطه‌ی انتهایی ربات (پنجه‌ی بازوی رباتیک) تنها از یک مسیر به زمین متصل شده باشد. اما مکانیزم موازی از چند مسیر نقطه‌ی انتهایی ربات را کنترل می کند.

در این قسمت می خواهیم به ساخت و برنامه نویسی یک بازوی رباتیک دو درجه آزادی بپردازیم. درجه‌ی آزادی یک مکانیزم عبارتست از تعداد مختصات مستقل برای توصیف موقعیت تمامی نقاط آن مکانیزم. به بیان دیگر درجه‌ی آزادی به تعداد حرکات مختلفی که یک مکانیزم در طول راستای مختلف و حول محورهای مختلف انجام می دهد گفته می شود.

بازوی مورد بحث در این قسمت از نوع مکانیزم سری و دارای دو درجه آزادی می‌باشد. همچنین در انتهای آن یک گریپر^۴ وجود دارد که برای برداشتن و گذاشتن اجسام استفاده می‌شود. در نتیجه در این ربات از ۳ موتور یا عملگر استفاده شده است. همچنین برای کنترل و توقف هر عضو، از دو میکروسوییچ استفاده شده است. همانطور که در قسمت ربات تفکیک رنگ گفته شد، برای بازگرداندن ربات به موقعیت اولیه از میکروسوییچ استفاده می‌شود. در این بازوی رباتیک از دو نوع سنسور به عنوان میکروسوییچ استفاده می‌نماییم. یک سنسور تماس (مانند ربات تفکیک رنگ) و یک سنسور نور. در سنسور نور می‌توان با توجه به نور دریافت شده از سنسور مقدار نزدیک بودن به موقعیت اولیه را بدست آورد. همچنین می‌توان با تشخیص رنگ، نزدیک شدن به موقعیت اولیه را تشخیص داد.

ابتدا با توجه به دستورالعمل موجود در پیوست انتهای کتاب، بازوی رباتیک را بسازید. همانطور که در حین ساخت این ربات مشاهده می‌کنید، دو سنسور تماس و نور به گونه‌ای بر روی آن قرار گرفته‌اند تا در موقعیت خاصی، موقعیت ربات را تشخیص داده و از حرکت بیشتر آن جلوگیری کنند. در روند برنامه‌نویسی با استفاده از این دو سنسور حرکات ربات را محدود کرده و در ابتدای کار، ربات را به موقعیت اولیه باز می‌گردانیم. در قدم اول باید مشخص کنیم که هر یک از اعضای ربات، در چه حالتی به شرایط اولیه می‌رسند. اگر محور مختصات منطبق بر بازو را به شکل زیر نشان دهیم، ربات ما حول دو راستای Z و Y دوران می‌کند. اگر بخواهیم دقیق‌تر مشخص کنیم، موتور C حول محور Z و موتور B حول محور Y دوران می‌نماید.



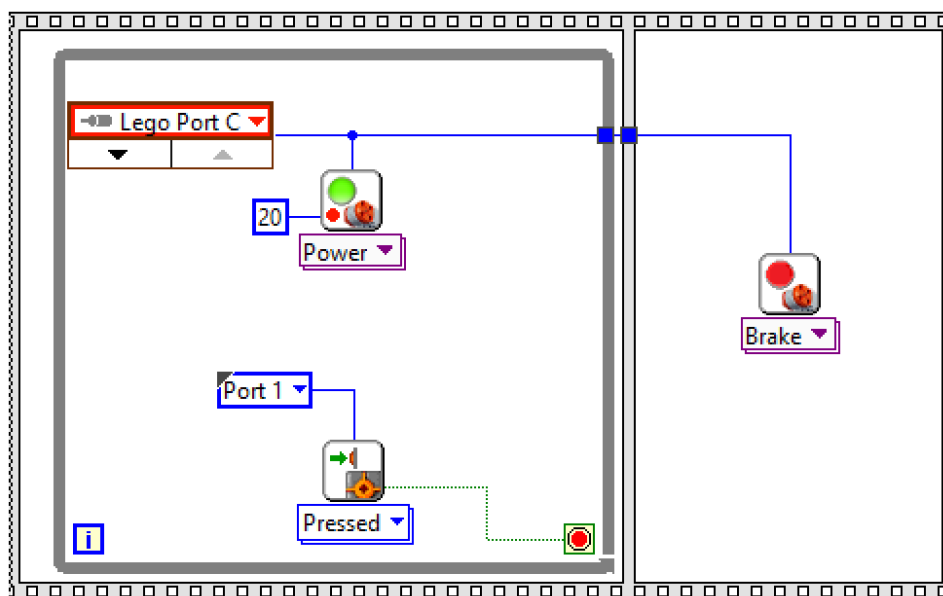
شکل ۱۸ محورهای حرکتی بازوی رباتیک

با دقت به ساختار ربات، با چرخش موتور C، در موقعیتی خاص، سنسور تماس فشرده می‌شود که بیانگر رسیدن این موتور به موقعیت اولیه است. همچنین با چرخش موتور A، لینک متصل به آن به سنسور نور نزدیک می‌شود که با استفاده از خروجی سنسور، می‌توان موقعیت اولیه این موتور را مشخص نمود. پس در قدم اول به طراحی برنامه‌ای برای بازگرداندن ربات به موقعیت اولیه و محدود کردن حرکات آن بوسیله‌ی سنسورها می‌پردازیم.

کنترل چرخش حول محور Z

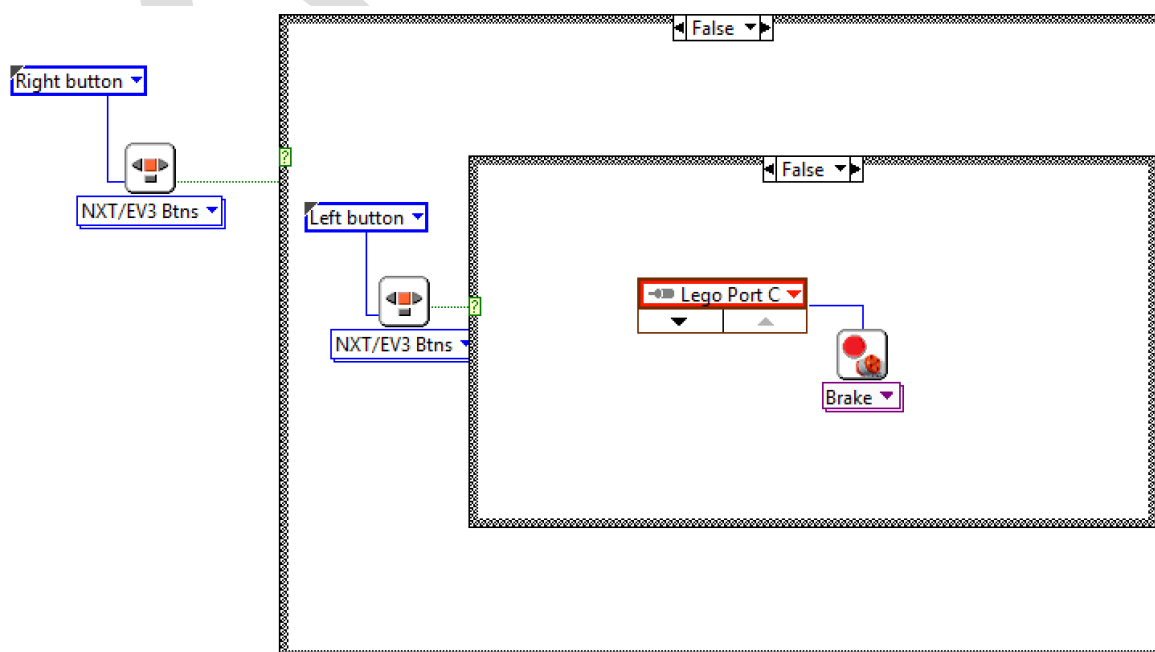
برای شروع برنامه‌نویسی این ربات، از موتور C شروع می‌نماییم. می‌دانیم که با چرخش موتور C لینک دوم و همچنین گریپر ربات حول محور Z دوران می‌کنند. در اینصورت با چرخش ربات C، زبانه‌ای که در مسیر سنسور تماس قرار دارد، سبب فشرده شدن دکمه‌ی سنسور تماس می‌شود و حرکت موتور متوقف خواهد شد. برای اینکه ربات همواره از این موقعیت شروع کند از اطلاعات بدست آمده از سنسور تماس استفاده می‌نماییم. همچنین در صورتی که ربات بعد از فشردن سنسور تماس نیز به حرکت خود ادامه دهد سبب آسیب رسیدن به موتور و سایر قطعات نیز خواهد شد. پس لازم است تا در صورت فشرده شدن سنسور تماس، حرکت موتور را متوقف نماییم و اجازه‌ی حرکت بیشتر را ندهیم.

برای بازگرداندن ربات به موقعیت اولیه باید موتور C را در جهت مناسب حرکت دهیم تا سنسور تماس فشرده شود. بعد از فشرده شدن سنسور تماس، حرکت موتور را متوقف می‌نماییم. با اجرای این بخش از برنامه در ابتدای فرآیند شروع کار ربات، ربات را از هر موقعیتی به موقعیت اولیه باز می‌گردانیم.



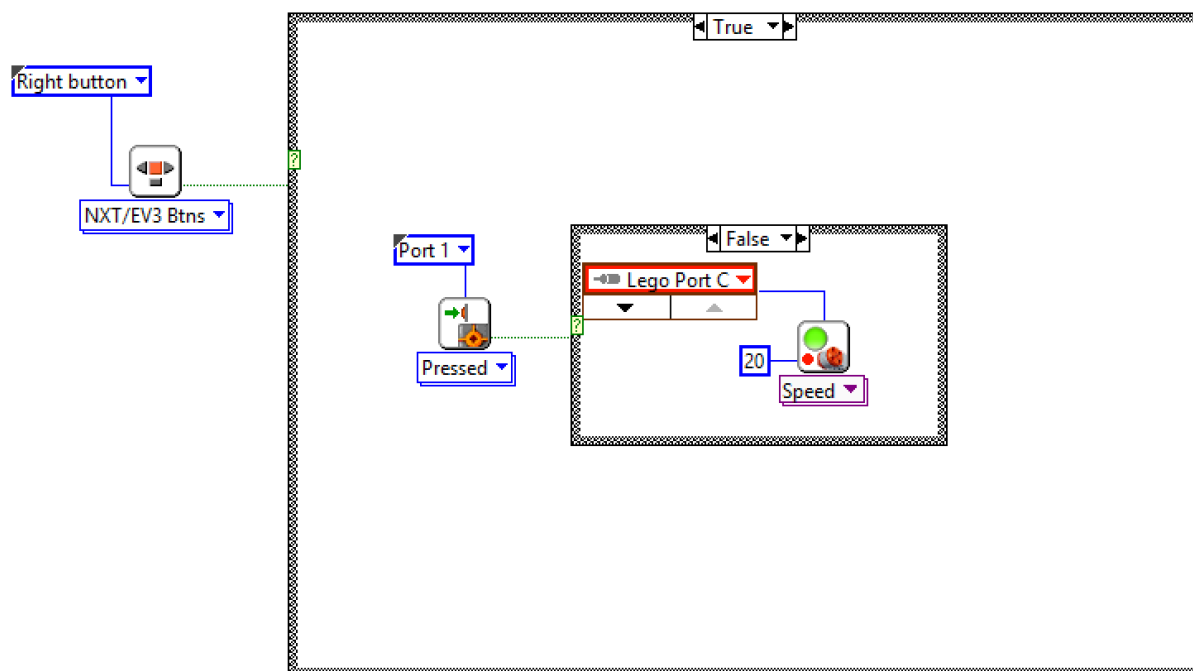
شکل ۱۹ بازگشت موتور C به موقعیت اولیه

حال به قسمت کنترل موتور C می‌رسیم. فرض کنید برای حرکت دادن ربات از کلیدهای بر روی بریک استفاده کنیم. در اینصورت برای کنترل موتور C نیاز به دو کلید هست تا آنرا در دو جهت حرکت دهد. برای مثال از دو کلید چپ و راست برای حرکت دادن این موتور استفاده می‌کنیم. در این صورت طبق برنامه‌هایی که در بخش‌های پیش نوشته شد، می‌توان به صورت زیر از این دو کلید برای حرکت موتور C استفاده کرد. در صورتی که هیچکدام از دکمه‌های چپ و راست زده نشود، موتور C ساکن خواهد ماند:



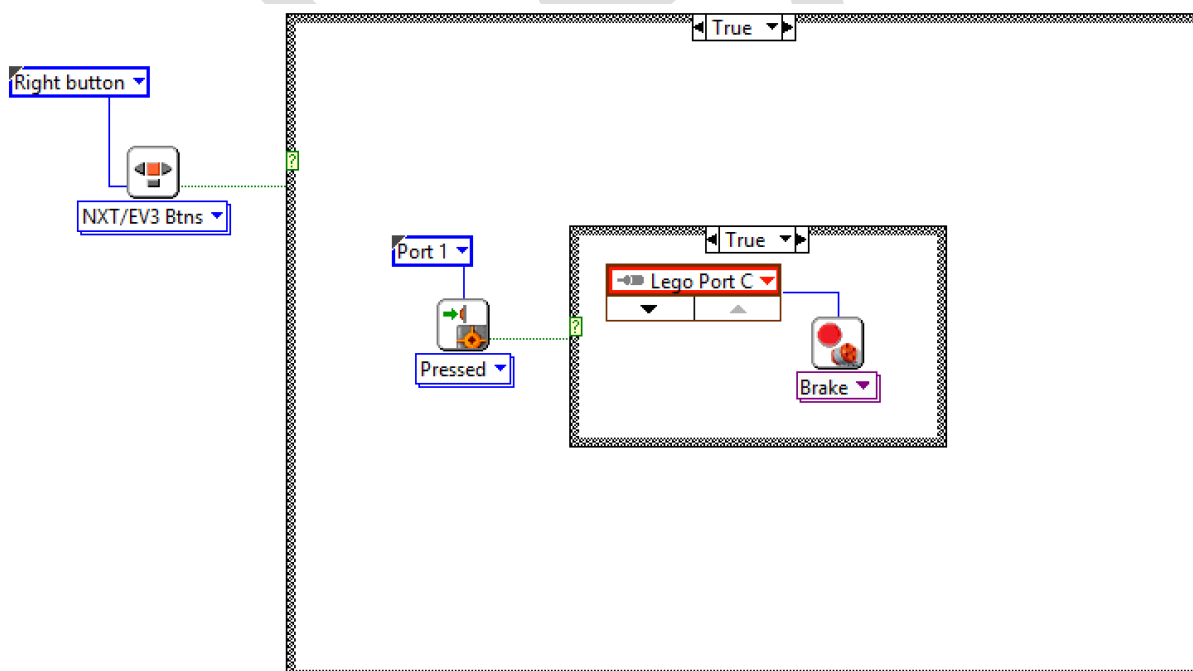
شکل ۲۰ حرکت موتور C با کلیدهای روی بریک

در صورتی که کلید راست فشرده شود، در صورتی که سنسور تماس فعال نشده باشد (موتور C به انتهای فضای کاری خود نرسیده باشد)، موتور به صورت ساعتگرد حرکت خواهد کرد:



شکل ۲۱ حرکت موتور C به سمت سنسور تماس با فشردن کلید راست

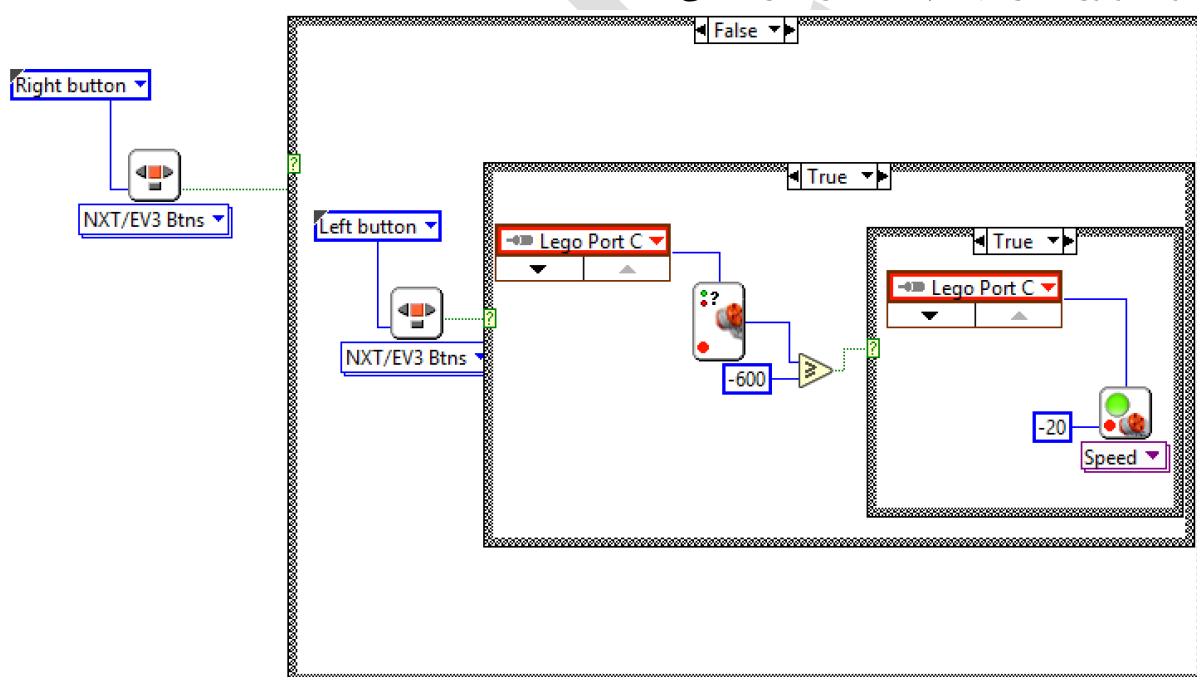
و در صورتیکه سنسور تماس فشرده شده باشد، موتور C متوقف شده و دیگر در این جهت حرکت نخواهد کرد:



شکل ۲۲ توقف موتور C در صورت رسیدن به سنسور تماس

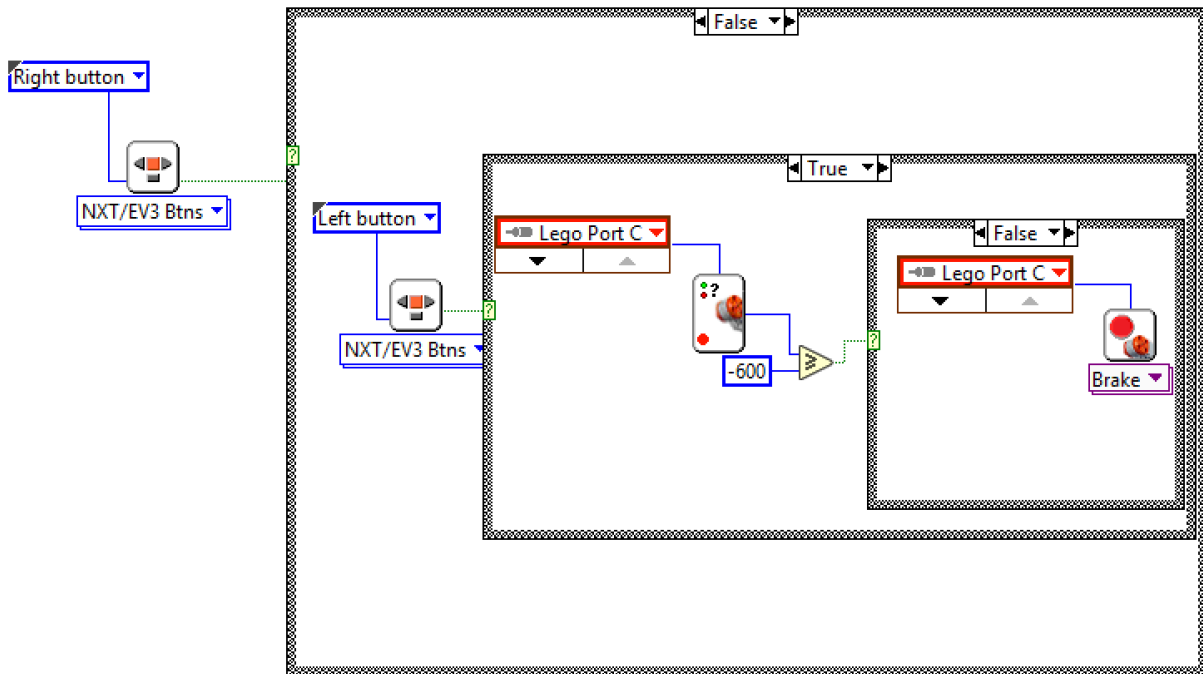
اما اگر دکمه‌ی چپ فشرده شود، موتور C در جهت پاد ساعتگرد شروع به حرکت خواهد کرد. در این قسمت نکته‌ای وجود دارد که بهتر از با دقت بررسی شود. همانطور که از ساختار ربات مشخص است، موتور C تا حدی می‌تواند از سنسور تماس دور شود، در نتیجه اگر این موتور بیش از حد چرخش نماید ممکن است اجزا

و اتصالات ربات را درگیر کرده و به آنها آسیب برساند. با توجه به این نکته، اگر فرض کنیم حد نهایی چرخش ربات را معادل نیم دور چرخش حول محور Z در نظر بگیریم، باید مقدار چرخش موتور C برای رسیدن بازو به این موقعیت را پیدا نماییم. با استفاده از انکودر موجود در هر موتور می توان موقعیت آنرا بدست آورد. برای این منظور از بلوک Motor Status استفاده می نماییم. همچنین برای اینکه همیشه موقعیت اولیه موتور C (موقعیتی که در آن سنسور تماس فشرده می شود) را مرجع در نظر بگیریم، بعد از رساندن موتور C به موقعیت اولیه، انکودر را با استفاده از بلوک Reset Encoders ریست کرده تا این موقعیت را با عدد صفر نشان دهد. در این صورت با چرخش موتور C، می توانیم با خواندن مقدار انکودر، فاصله از موقعیت اولیه را بدست آوریم. اگر حد نهایی چرخش موتور C در جهت پاد ساعتگرد را ۶۰۰ درجه در نظر بگیریم (تقریباً نیم دور فضای حرکت بازوی ربات)، با فشردن دکمه چپ، تا هنگامی که عدد خوانده شده از انکودر از مقدار ۶۰۰ درجه بیشتر نشود، موتور C در جهت پادساعتگرد حرکت می کند:



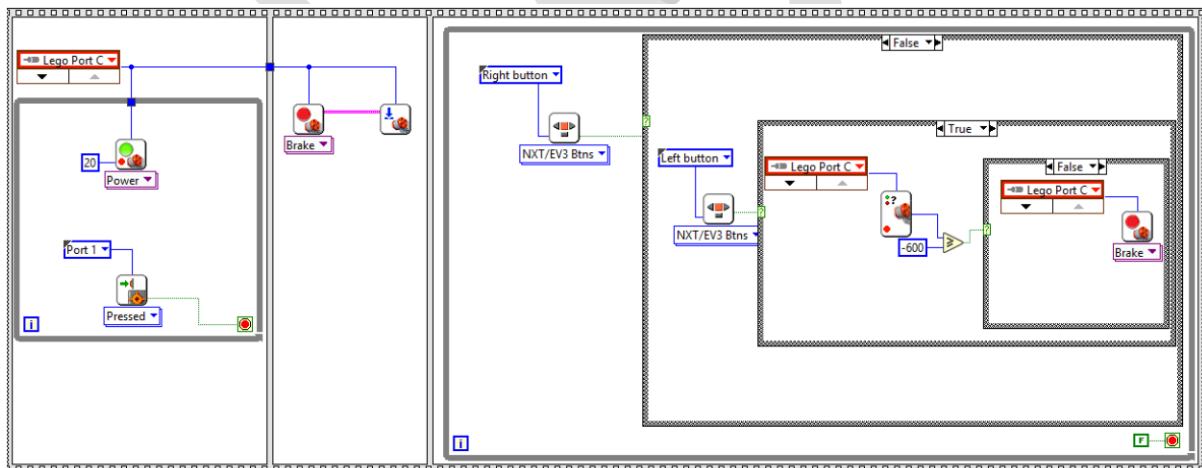
شکل ۲۳ حرکت موتور C تا فاصله‌ی ۶۰۰ درجه دورتر از سنسور تماس

و در صورتی که مقدار انکودر موتور از ۶۰۰ درجه عبور کند، موتور متوقف خواهد شد:



شکل ۲۴ توقف موتور C در صورت رسیدن به مقدار ۶۰۰ درجه

حال که بخش کنترل برنامه نیز تکمیل شد، لازم است تا آنرا در یک حلقه‌ی بینهایت قرار داده و در ادامه‌ی برنامه‌ی قبل که موتور C را به موقعیت اولیه برمی‌گرداند، قرار دهیم. همچنین فراموش نکنیم که بعد از رسیدن به موقعیت اولیه، لازم است تا مقدار انکودر موتور، ریست شود:

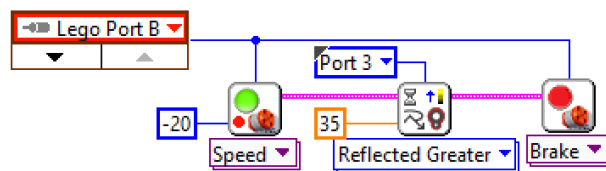


شکل ۲۵ تکمیل برنامه با قرار دادن در حلقه بینهایت

با اجرای این برنامه، در صورتی که موارد گفته شده رعایت شده باشد، ابتدا موتور به موقعیت اولیه باز گشته و سپس با فشردن دکمه‌های چپ و راست، می‌توان موتور C را کنترل نمود. فراموش نکنید تا این برنامه را با نام مناسب و در مسیر مشخص، ذخیره نمایید تا در ادامه از این برنامه استفاده نماییم. برای مثال ما در اینجا این برنامه را با نام arm-z-axis.vi در مسیر مشخص، ذخیره می‌نماییم.

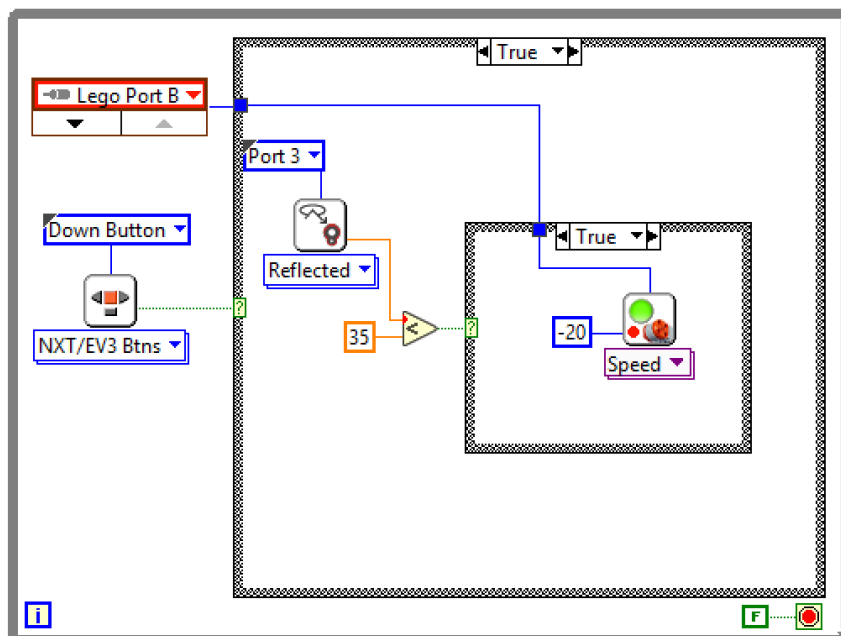
چرخش حول محور Y

چرخش حول محور Y تا حدی مشابه چرخش حول محور Z می‌باشد با این تفاوت که در این محور به جای سنسور تماس، از سنسور نور به عنوان میکروسویچ استفاده می‌گردد. مانند قبل، ابتدا برنامه‌ی حرکت موتور محور Y یعنی موتور B را به موقعیت اولیه طراحی می‌نماییم و سپس به کنترل آن می‌پردازیم. برای رساندن موتور B به موقعیت اولیه باید این موتور را در جهت مناسب تا جایی حرکت داد تا زبانه‌ی روبروی سنسور نور، به چشمی سنسور نزدیک شده و سنسور آنرا ببیند. برای اینکار از مد Reflected در سنسور نور استفاده کرده تا به این طریق نزدیکی زبانه به چشمی را بخوانیم. با کمی بررسی می‌توان فهمید که هر گاه زبانه به چشمی نزدیک شود، مقدار خوانده شده در مد Reflected تقریباً بیشتر از مقدار ۳۵ خواهد بود. در نتیجه باید موتور را آنقدر به سنسور نزدیک کنیم تا مقدار خوانده شده از ۳۵ بیشتر شود.



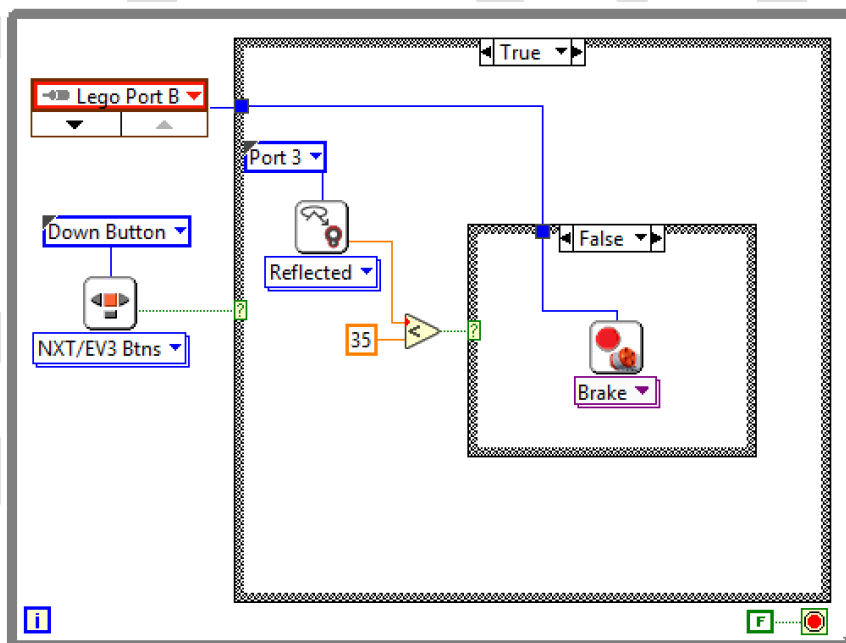
شکل ۲۶ خواندن مقدار سنسور نور برای توقف موتور B

با اجرای این برنامه موتور B، بازو را به سمت بالا حرکت داده و هنگامی که به بالاترین حد خود (نزدیک شدن زبانه به سنسور نور) برسد، آنرا متوقف می‌کند. دقت کنید که در صورتی که سرعت موتور زیاد باشد ممکن است سبب آسیب رسیدن یا عملکرد نادرست این برنامه شود پس نباید سرعت موتور را عدد بزرگی انتخاب نمود. در اینجا سرعت موتور را مقدار ۲۰ و در جهت منفی در نظر گرفته‌ایم. همچنین دقت کنید که سنسور نور به پورت ۳ متصل شده است و باید در بلوک Sensor، شماره پورت متصل به سنسور نور را مشخص کرد. در مرحله بعد می‌خواهیم برنامه‌ای بنویسیم که با کلیدهای بالا و پایین بر روی بریک، بتوان حرکت موتور B را کنترل نمود. همچنین باید محدودیت‌های فیزیکی بازو را نیز در نظر گرفت. با توجه به جهت قرارگیری بریک نسبت به ربات، دکمه‌ی پایین بلوک را برای حرکت موتور B به سمت سنسور نور و دکمه‌ی بالای بریک را برای حرکت عکس آن انتخاب می‌کنیم. با دو ساختار Case Structure درون هم، می‌توانیم این رفتار را ایجاد کنیم. برای حالتی که دکمه‌ی پایین فشرده شده و هنوز بازو به نزدیکی سنسور نور نرسیده است داریم:



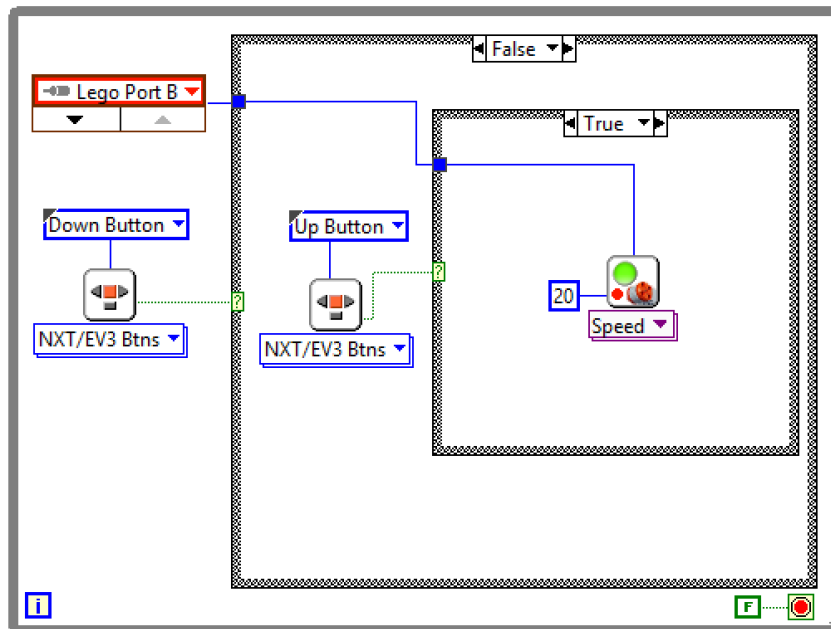
شکل ۲۷ حرکت موتور B به سمت سنسور نور

اگر بازو به انتهای حرکت خود (نزدیک شدن زبانه به سنسور نور) برسد، آنگاه باید موتور B را متوقف کرد:



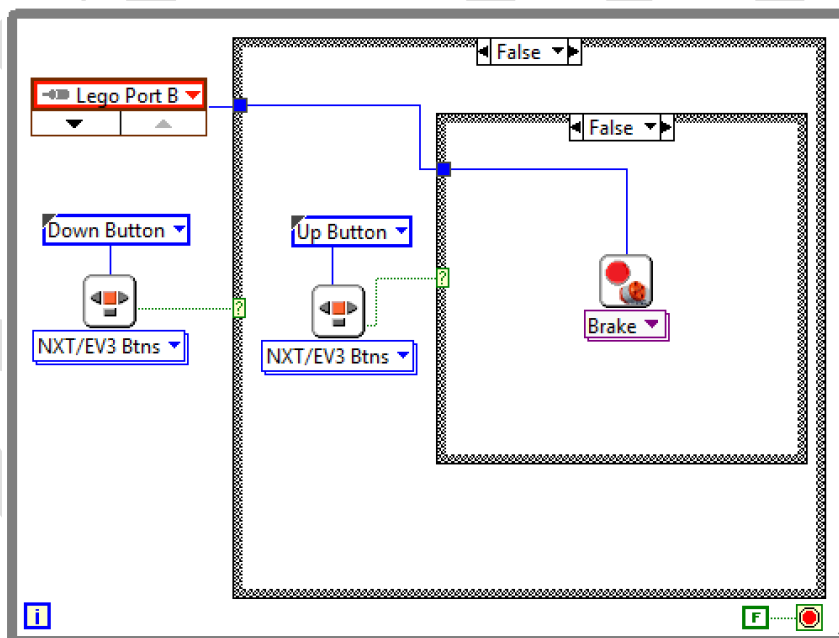
شکل ۲۸ توقف موتور B در صورت رسیدن به سنسور نور

برای مواردی که دکمه‌ی بالا فشرده می‌شود (در نتیجه دکمه‌ی پایین فشرده نشده است)، موتور در جهت مثبت حرکت می‌کند:



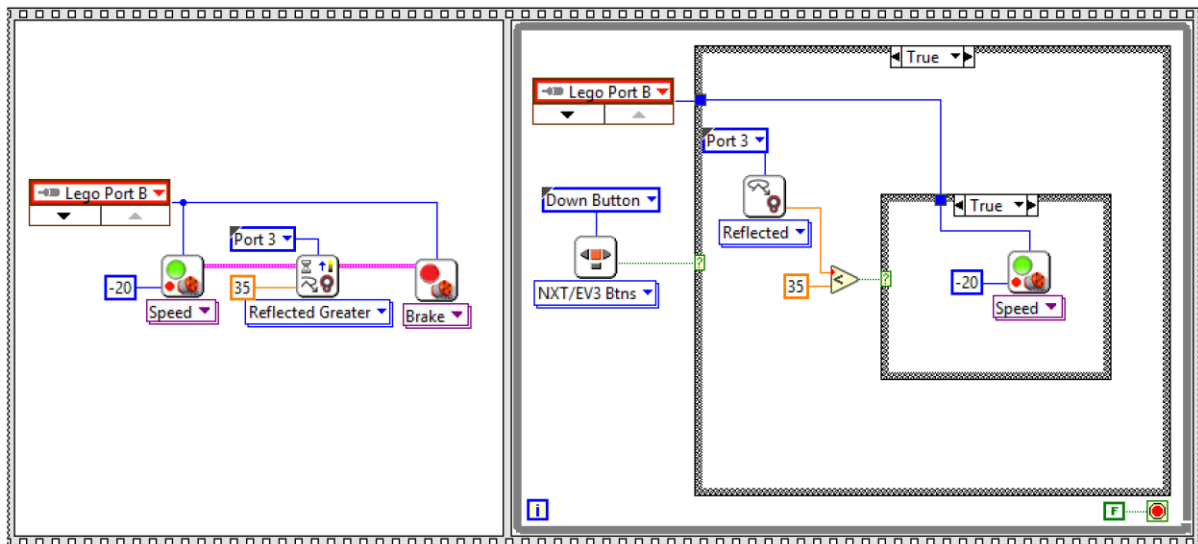
شکل ۲۹ دور شدن موتور B از سنسور نور

در صورتی که هیچکدام از دکمه‌های بالا و پایین فشرده نشوند، موتور باید متوقف شود:



شکل ۳۰ توقف موتور B در صورت فشرده نشدن کلیدهای بالا و پایین

در نهایت برای کنترل و حرکت محور Y از Flat Sequence استفاده کرده و قسمت اول و دوم برنامه را به ترتیب کنار هم قرار می‌دهیم:



شکل ۳۱ تکمیل حرکت موتور B مربوط به حرکت محور Y

فراموش نکنید این فایل را با نام مناسب (مثلا arm-y-axis) ذخیره نمایید تا در ادامه از آن استفاده کنیم.

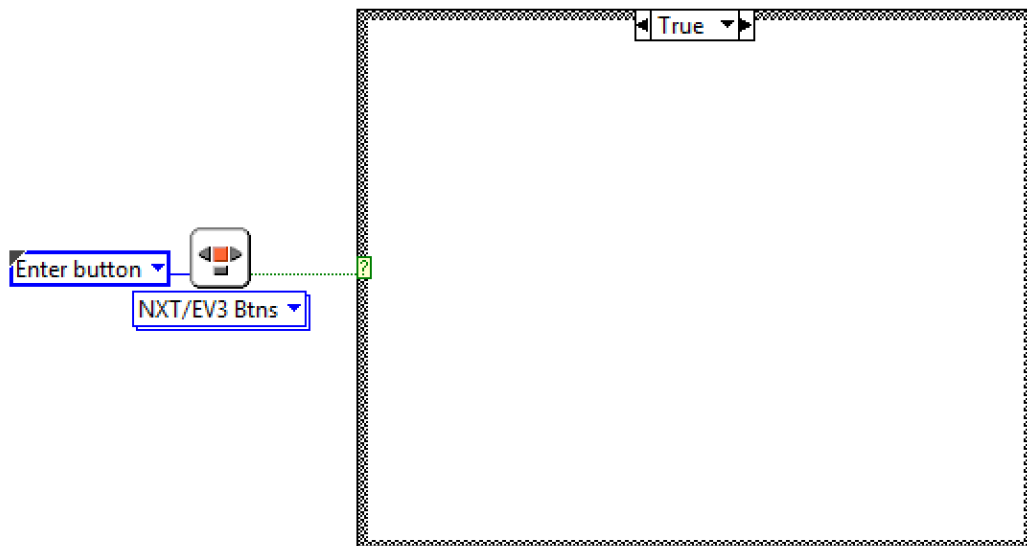
حرکت گریپر

حال که حرکات بازو حول دو محور Y و Z طراحی گردید لازم است تا گریپر به گونه‌ای عمل کند تا وظیفه‌ی برداشتن و گذاشتن^۵ اجسام را به خوبی انجام دهد. از گریپر یا مکانیزم‌های مشابه به طور گسترده در بازوهای رباتیک صنعتی در کارخانه‌ها و خطوط تولید استفاده می‌گردد. مکانیزم گریپر می‌تواند به شکل‌های مختلفی باشد اما گریپری که در اینجا استفاده شده است با چرخش یک موتور، دو فک گریپر را به یکدیگر نزدیک کرده و جسمی را که در آن بین قرار داشته باشد می‌گیرد.

با دقت به مکانیزم گریپر متوجه خواهیم شد که برای هر بار باز و بسته شدن آن لازم است تا جهت حرکت موتور تغییر کند. یعنی با فرض بسته بودن گریپر در حالت اولیه، با چرخش تقریباً ۱۸۰ درجه‌ای موتور A گریپر ابتدا باز شده و سپس دوباره بسته خواهد شد. همچنین از آنجایی که نمی‌توانیم موقعیتی را برای باز یا بسته بودن انکودر گریپر مشخص کنیم (زیرا با توجه به ابعاد مختلف اجسام گرفته شده، این موقعیت تغییر خواهد کرد) پس به سختی می‌توان عددی را به عنوان موقعیت باز یا بسته‌ی گریپر پیدا نمود. بهترین روش آن است که در هنگامی که لازم است تا گریپر جسمی را بگیرد، آنرا در مدت زمان مشخصی (مثلاً ۱ ثانیه) در جهت مشخص (تغییر جهت چرخش با هر بار حرکت گریپر) و با قدرت مشخص دوران دهیم.

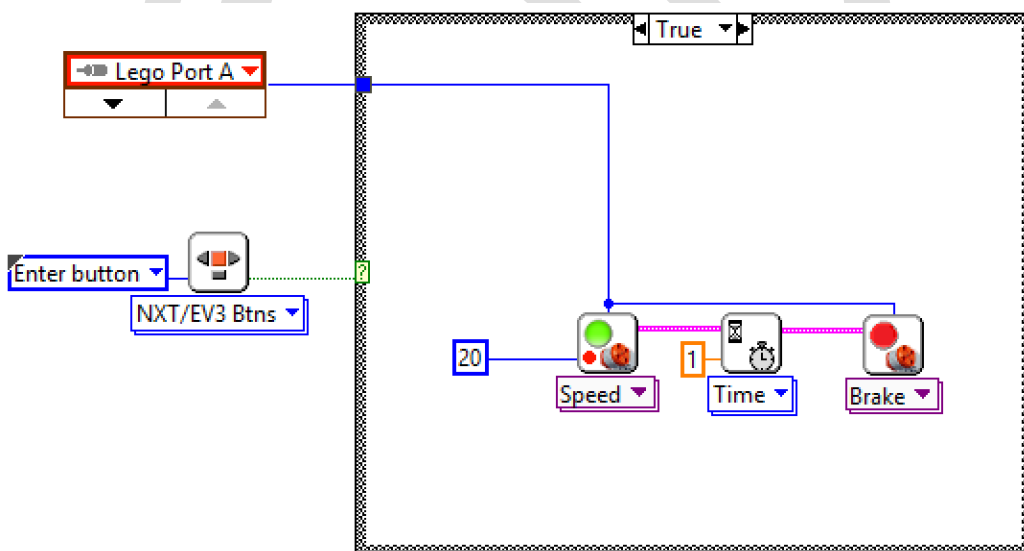
فرض می‌کنیم می‌خواهیم با هر بار فشردن کلید وسط روی بریک، گریپر باز و بسته شده و در نتیجه جسمی را بگیرد. پس در یک VI جدید، یک Case Structure با شرط کلید وسط می‌سازیم:

^۵ Pick and Place



شکل ۳۲ خواندن مقدار کلید وسط بزرگ

همانطور که گفته شد با هر بار فشردن کلید وسط، باید موتور A به مدت زمان ۱ ثانیه در جهت مناسب حرکت کند.



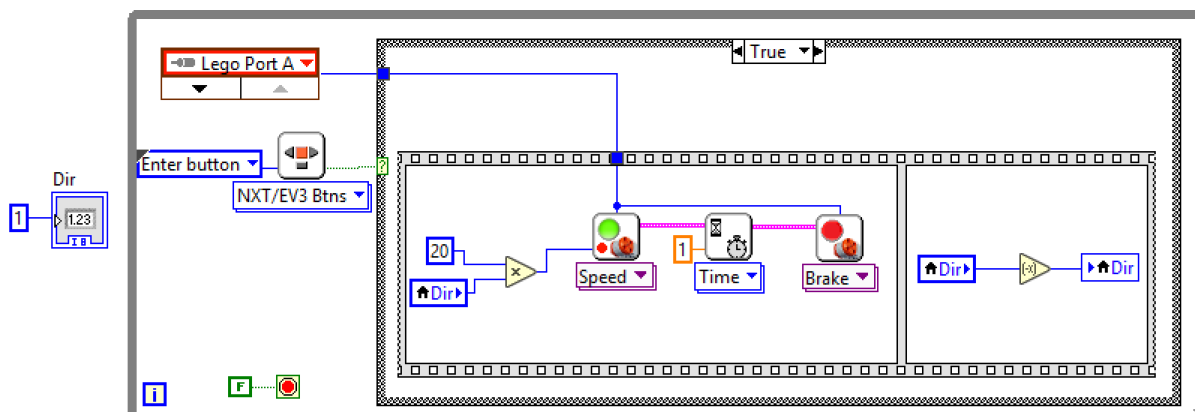
شکل ۳۳ حرکت موتور Gripper در جهت مناسب

اما طبق نکته‌ی گفته شده، موتور نمی‌تواند همیشه در یک جهت حرکت کرده و گریپر را باز و بسته نماید. پس لازم است تا بعد از هر بار فشرده شدن دکمه وسط، جهت حرکت موتور تغییر داده شود تا در دفعه‌ی بعد نیز حرکت موتور، گریپر را باز و بسته کند. برای اینکار از یک متغیر به نام Dir استفاده می‌کنیم. به این صورت که ابتدا مقدار اولیه ۱ را در این متغیر قرار داده و با هر بار حرکت موتور، یک علامت منفی در این متغیر ضرب می‌کنیم. در نتیجه اگر از این متغیر جهت تعیین چرخش موتور استفاده کنیم، با هر بار فشردن دکمه، جهت موتور تغییر خواهد کرد.

برای تعریف یک متغیر، همانطور که در بخش‌های قبل نیز استفاده شد، در پنجره Front Panel کلیک راست کرده و از بخش Numeric، یک Numeric Indicator را در برنامه خود قرار دهید. سپس نام این شاخص

عددی را به "Dir" تغییر دهید. برای نوشتن در این متغیر در پنجره Block Diagram، بر روی بلوک مربوط به شاخص عددی کلیک راست کرده و از منوی نشان داده شده، گزینه Create → Local Variable را انتخاب کنید. در صورتی که بخواهید مقدار موجود در داخل این متغیر را بخوانید بر روی بلوک Local Variable ساخته شده کلیک راست کرده و Change to read را انتخاب کنید.

پس برای اینکه جهت موتور در هر بار چرخش تغییر کند، بعد از اینکه کلید وسط فشرده شد و موتور B را حرکت دادیم، علامت متغیر "Dir" را عوض می‌کنیم. در نتیجه برنامه ما در حال حاضر به شکل زیر می‌باشد.



شکل ۳۴ تغییر چرخش حرکت گریپر با هربار فشردن کلید وسط

دقت کنید که قبل از اینکه برنامه وارد حلقه while شود، مقدار ۱ را به متغیر Dir اختصاص دادیم. در نهایت این برنامه را با نام مناسب (در اینجا با نام "arm-gripper") ذخیره می‌نماییم.

برنامه نهایی بازوی رباتیک

تا به اینجا، سه درجه‌ی آزادی بازوی رباتیک را برنامه‌نویسی نمودیم اما هر کدام از حرکات ربات به طور جداگانه و مستقل برنامه ریزی شده است. یعنی به طور همزمان نمی‌توان حرکات مختلف را به بازو فرمان داد. از آنجایی که هر کدام از ۳ برنامه‌ی طراحی شده در قسمت‌های اخیر، به طور مستقل و کامل می‌باشند، تنها لازم است تا آنها را همزمان با یکدیگر اجرا کنیم. چون بریک در آن واحد تنها قادر به اجرای یک برنامه می‌باشد، باید برنامه‌ی دیگری را در لبویو طراحی نمود تا هر ۳ حرکت ربات را در برگیرد. برای اینکار یک VI جدید ایجاد کنید (فشردن کلید Ctrl+N در پنجره برنامه لبویو). برای وارد کردن یک VI به برنامه خود، در پنجره Block Diagram کلیک راست کرده و از منوی Functions، گزینه Select a VI... را انتخاب نمایید. در پنجره‌ی باز شده، باید فایل VI مورد نظر را انتخاب کنید. به ترتیب ۳ برنامه ساخته شده در قسمت‌های قبل را که ما با نام "arm-y-axis"، "arm-z-axis" و "arm-gripper" ذخیره نمودیم به برنامه جدید خود اضافه می‌کنیم.



شکل ۳۵ افزودن VI‌های ساخته شده در VI جدید

چون در هر یک از VI‌ها، حلقه while بینهایت وجود دارد پس نیازی به وجود حلقه while در برنامه‌ی نهایی نمی‌باشد. حال با آپلود و اجرای این برنامه بر روی ربات خود ابتدا محورهای ربات به موقعیت اولیه بازگشته (به اصطلاح home می‌شوند) و سپس می‌توانید با کلیدهای چپ و راست، ربات را حول محور Y و کلیدهای بالا و پایین، ربات را حول محور Z دوران دهید. همچنین با فشردن کلید وسط بریک، گریپر باز و بسته می‌شود و اگر جسمی در آن موقعیت قرار داشته باشد، گریپر آنرا می‌گیرد.

تمرین بازوی رباتیک

۱. بازوی رباتیک را به گونه‌ای برنامه‌نویسی کنید تا در یه نقطه مشخص و اجسام را برداشته و با مسیر مشخص، آنرا به نقطه‌ای دیگر منتقل کند.
۲. برنامه‌های طراحی شده را با گونه‌ای تغییر دهید تا بتوان بازوی رباتیک را با استفاده از Front Panel کنترل نمود.