به نام خدا

دانشگاه تهران

دانشکده مهندسی مکانیک

طراحی و پیاده‌سازی رابط انسان و ماشین برای دستگاه‌های اتوماسیون جهت مونتاژ قلم انسولین

نام و نام خانوادگی	**بهیاد زرنقی**
شماره دانشجویی	**۸۱۰۶۹۸۲۴۹**
استاد راهنما	**دکتر علیرضا صادقی**
تاریخ ارسال گزارش	**۱۴۰۲/۱۱/۱۲**

چکیده

در این گزارش روند کلی از تهیه و اجرای یک رابط کاربری برای مکانیزم اتوماسیونی تحت عنوان رابط انسان و ماشین جهت مونتاژ یکی از قطعات اصلی قلم انسولین ارائه شده است. کلیات پروژه در قالب مقدمه و تاریخچه‌ی HMI و مراحل طی شده جهت اتمام پروژه شرح داده شده است. در ادامه روند اجرا و طراحی پروژه ارائه شده که شامل دو بخش توضیحات مکانیزم و توضیحات طراحی رابط کاربری مد نظر به صورت مرحله‌ای می‌باشد. بخش اصلی پروژه که با محوریت ساخت و طراحی رابط کاربری می‌باشد، خود در دو مرحله‌ی رابط اطلاعاتی و رابط گرافیکی تعریف شده است. رابط اطلاعاتی به کمک ماژول Flask انجام خواهد پذیرفت و بخش گرافیکی نیز با کمک برنامه‌نویسی HTML و CSS تهیه خواهد شد. در نهایت نحوه‌ی فعالسازی و اجرای این رابط کاربری برای سیستم‌های مونتاژ بر روی کامپیوترهای شخصی توضیح داده شده است.

کلمات کلیدی: رابط کاربری – مونتاژ – HMI

فهرست

شکل‌ها

جدول‌ها

کلیات پروژه

۱-۱. مقدمه

امروزه در صنعت ساخت و تولید، سیستم‌های اتوماسیون برای پیشبرد فرآیندهای مونتاژی به صورت بهینه و با سرعت بالا نقش موثری دارند. با وجود پیشرفت‌های متعدد در این زمینه، همچنان نیازمند این است که عملکرد این سیستم‌ها با اضافه شدن بعضی از ویژگی‌های جدید ارتقا یابد و فرآیندها به صورت هوشمندتری جلو برده شوند. از مواردی که می‌تواند منجر به تحقق این مهم گردد، اضافه شدن رابط کاربری HMI مناسب برای این سیستم‌ها می‌باشد. این رابط کاربری به طور خلاصه جهت مانیتور کردن اطلاعات و پارامترها و همچنین اعمال فرمان به کنترل کننده‌های صنعتی اتوماسیون مورد استفاده قرار خواهد گرفت. همچنین با کمک این رابط، امکان پایش دائم در سیستم‌های مختلف میسر می‌شود و اپراتور می‌تواند به کمک این ابزار در صورت بروز هر گونه ایراد و اشکال در سیستم، آن را سریعاً دریافته و با انجام اقدامات ضروری و مورد نیاز سیستم را متوقف نماید تا از بروز مشکلات بعدی جلوگیری نماید. بنابرین این رابط کاربری بایستی بتواند سیستم را در شرایط متفاوت کنترل نماید. برای اجرای این پروژه به شکل مناسب هماهنگی دقیقی بین سخت‌افزار و نرم‌افزار خط اتوماسیون باید برقرار گردد و این مسئله از چالش‌های این پروژه خواهد بود.

۱-۲. تاریخچه

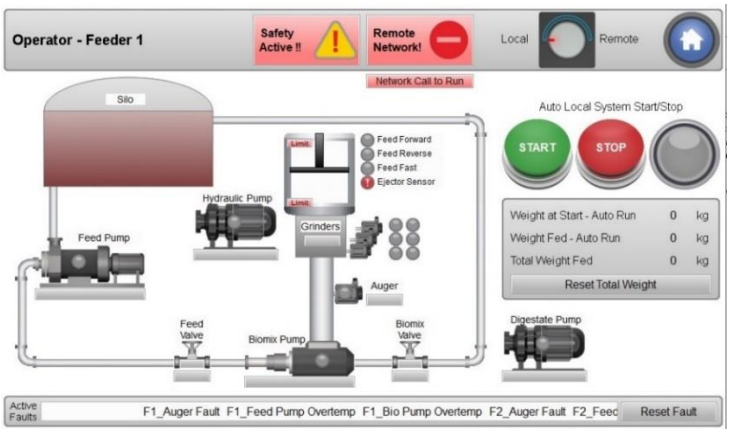
تاریخچه HMI شامل رابط دسته‌ای (۱۹۶۸- ۱۹۴۵)، رابط کاربری خط فرمان (۱۹۶۹) و رابط کاربری گرافیکی (۱۹۸۱) می شود. رابط دسته ای یک رابط کاربری غیر تعاملی است که در آن کاربر جزئیات فرایند کار دسته ای را مشخص می‌کند. سپس در زمانی که کلیه پردازش‌ها انجام می گردد، خروجی ارائه می‌شود. رابط خط فرمان مکانیزمی است که با تایپ دستورات برای انجام وظایف خاص با یک سیستم عامل یا نرم افزار کامپیوتری در تعامل است. مفهوم رابط خط فرمان زمانی رواج پیدا کرد که ماشین‌های تله تایپ نویس در دهه ۱۹۵۰ به رایانه‌ها متصل شدند و در صورت تقاضا به ارائه نتایج پرداختند. به مرور زمان همگام با پیچیده شدن رابط‌ها، بهره‌گیری از رابط‌های گرافیکی آسان شد. یکی از رابط‌های کاربردی، رابط کاربری گرافیکی (GUI) است که به افراد اجازه می‌دهد تا با برنامه‌ها به شیوه‌ای بصری ارتباط برقرار کنند. به طور مثال با رابط کاربری گرافیکی می توان به جای دستورات متنی از نمادهای روی صفحه های لمسی استفاده کرد. HMI مدرن به صورت مستقیم از رابط کاربری گرافیکی بهره می‌گیرد و به کنترل و کارکرد بسیار موثرتر از ماشین آلات منجر می‌شود. از مهم‌ترین مزیت‌های HMI ها نسبت به

همتایان خود یعنی PLC ها، قابلیت پایش و دریافت داده از سیستم به صورت زمان واقعی یا به اصطلاح Real Time می‌باشد. به این صورت که می‌تواند در هر لحظه وضعیت سیستم را ملاحظه کرده و در صورت بروز ایراد یا اختلال در روند اتوماسیون این مشکل را سریعاً بر طرف نماید.[1]

۱-۳. رابط انسان با ماشین (HMI)

به طور کلی رابطه‌ی انسان با ماشین به عنوان یک مسیر ارتباطی بین یک دستگاه خاص و کاربر تعریف می‌شود. این رابط به صورت یک ویژگی یا یک برنامه‌ی نرم‌افزاری خواهد بود که انسان را قادر می‌سازد تا با استفاده از آن با ماشین های مختلف تعامل برقرار کند. این ابزار در صنعت در مواردی متفاوتی می‌تواند استفاده شود تا کاربر بتواند از نحوه‌ی عملکرد سیستم مطلع شود و با استفاده از فرمان‌های تعریف شده در این رابط، دستورات لازم را برای تغییر روند سیستم و فعالسازی و یا غیرفعالسازی هر بخش را ارسال نماید.

به طور کلی رابط انسان با ماشین متشکل از دو بخش رابط گرافیکی و رابط اطلاعاتی خواهد بود. رابط گرافیکی همانطور که از نامش برآورد می‌شود، بخش بصری این رابط بین انسان و ماشین خواهد بود تا کاربر بتواند مطابق با آن با سیستم ارتباط داشته باشد. این بخش نیازمند یک پلتفرمی می‌باشد که کاربر و یا اپراتور به آسانی بتواند درک دقیق و کامل از روند یک ماشین را داشته باشد. برای همین منظور، طراحی این بخش نیازمند توجه به جزئیات متعددی خواهد بود تا این رابط گرافیکی را تا حد امکان کاربر پسند نماید. رابط اطلاعاتی در واقع پشت صحنه‌ی این تکنولوژی می‌باشد که وظیفه‌ی دریافت اطلاعات از سیستم و انتقال دادن این اطلاعات به صورت قابل نمایش به بخش گرافیکی رابط انسان با ماشین خواهد بود. روند انجام این کار متناسب با سیستم عامل‌های متفاوت می‌تواند تغییر نماید. نکته‌ی اصلی و حائز اهمیت در این بخش توانایی این بخش در تثبیت داده‌های مد نظر و انتقال همزمان این داده‌ها به صفحه‌ی نمایش سیستم خواهد بود تا کاربر بتواند در هر لحظه پایش صحیحی از سیستم داشته باشد. همچنین در وظایف این بخش بایستی مسئله‌ی انتقال فرمان‌های مورد نیاز تعریف گردد.



شکل ۱. نمونه‌ای از یک رابط گرافیکی استفاده شده برای سیستم یک نیروگاه [۲]

۱-۴. رابط کاربری گرافیکی برای اتوماسیون قلم انسولین

قلم انسولین از جمله محصولات پرمتقاضی در طیف تجهیزات پزشکی می‌باشد که بایستی برای مقابله با دیابت به صورت مرتب توسط بیماران مصرف گردد. برای تولید انبوه این محصول، سیستم‌های اتوماسیون گسترش یافته‌اند تا قطعات تشکیل دهنده‌ی این محصول را به صورت اتوماتیک مونتاژ نمایند. ساختار سیستم‌های اتوماسیون به دلیل پیچیدگی‌های متعدد می‌تواند دچار خطاهایی بشود که روند مونتاژ را متوقف نماید. یا در مواردی علارغم عدم ایجاد توقف، منجر به مونتاژ غیر اصولی قطعات قلم گردد که باعث تولید قلم‌های معیوب خواهد شد. این حالت در اغلب موارد به دور از چشم اپراتور رخ خواهد داد و بایستی با استناد بر سنسورهای متنوع این مورد ثبت گردد و به نحوی به اپراتور این اطلاعات گزارش داده شود. به همین منظور است که الزام ایجاد یک رابط کاربری بین انسان و دستگاه به وجود می‌آید تا بازده فرآیند مونتاژ را تا حد امکان بالا ببرد و به تبع آن، دستگاه‌های مونتاژ بتوانند محصولات نهایی با کیفیت بهتر و نرخ تولید بالاتر را عرضه نمایند. بر اساس قابلیت اطلاع‌رسانی این رابط کاربری نسبت به نحوه‌ی عملکرد دستگاه مونتاژ، اپراتور می‌تواند به کمک آن، دستورات لازم برای به عمل‌آوری بخش‌های مختلف سیستم مونتاژ را فعال نماید. این مهم با استفاده از جزءهای موجود در بخش گرافیکی میسر خواهد بود تا با استفاده از آن دستورات را به بخش‌های متناظر دستگاه مونتاژ ارسال نماید.

شکل ۲. نمونه‌ای از قلم انسولین

شکل ۳ نمایی از سیستم اتوماسیون قلم انسولین را نمایش می‌دهد که به صورت ریل و واگن طراحی شده است که هر بخش از آن وظیفه‌ی اضافه نمودن یکی از قطعات قلم انسولین را دارا خواهد بود.

شکل ۳. مکانیزم استیج ۱ مونتاژ قلم انسولین

رابط کاربری گرافیکی طراحی شده در این پروژه به طور خاص برای بخشی از مکانیزم که قابلیت اجرایی به صورت دقیق دارد آماده شده است. این بخش مربوط به مونتاژ قطعه‌ی موسوم به چرخدنده‌ها خواهد بود که در بخش بعدی به صورت مفصل شرح داده شده است.

۱-۵. زمان‌بندی انجام پروژه

با توجه به پیچیدگی‌های موجود در اجرای پروژه، کلیت فرآیند به چند قسمت تقسیم شده و برای هر قسمت نیز هدف‌گذاری‌های تعریف گردیده است. در فاز اول ابزارهای اولیه برای ایجاد این رابط کاربری مورد تحقیق و بررسی قرار گرفته است. پس از تعیین نحوه‌ی انجام کار از بین راه‌حل های ارائه شده توسط تجربیات گذشته، یک نمونه‌ی ساده و اولیه‌ای از این رابط تحت یک وب سرور آنلاین ایجاد شده است. پس از اطمینان حاصل نمودن از صحت عملکرد نمونه‌ی اولیه، اینبار همین روند را برای یک مکانیزم آماده تست کردیم تا در صورت ایجاد محدودیت‌هایی، راه‌حل های مناسبی اتخاذ گردد. در فاز بعدی، بعد نهایی شدن قابلیت ارتباط مکانیزم‌های مونتاژ با رابط گرافیکی از نظر ارتباط و به اشتراک گذاشتن داده‌ها ، تمرکز اصلی بر روی بهبود بخش گرافیکی و بصری گذاشته شد تا با استفاده از المان‌های پیچیده‌تر و آماده، رابط کاربری گرافیکی کاربر پسندتر شود. در آخر نیز با تکمیل نواقص و ارتقای یکسری از ایرادات پروژه به اتمام رسید.

جدول ۱. زمانبندی مراحل انجام پروژه

تاریخ (ماه)	فعالیت
مهر ماه	تحقیق اولیه در مورد نحوه‌ی اجرا و راه‌اندازی وب سرور
آبان ماه	ساختن نمونه‌ای اولیه برای کنترل عملگرها و سنسورها
آذر ماه	طراحی HMI برای استیج‌های مختلف مونتاژ
دی ماه	افزودن المنت‌های گرافیکی رابط کاربری
بهمن ماه	تکمیل نواقص و ارائه‌ی نسخه‌ی نهایی

نحوه‌ی انجام پروژه

با توجه به توضیحات ارائه شده در مقدمه، محوریت پروژه بر اساس طراحی یک قالب رابط کاربری برای بخشی از مکانیزم اتوماسیون قلم انسولین می‌باشد. برای انجام این مهم بایستی در وهله‌ی اول، درک مناسبی از نحوه‌ی عملکرد سیستم اتوماسیون و به طور خاص بخشی از سیستم مونتاژ که می‌خواهیم رابط کاربری را برای آن راه‌اندازی کنیم را داشته باشیم. در ادامه با درک روابط و نیازهای موجود برای این بخش، به سراغ طراحی این رابط کاربری می‌رویم که خود متشکل از دو بخش رابط اطلاعاتی (Back-End) رابط گرافیکی (Front-End) خواهد بود که متناظراً وظیفه‌ی برقراری ارتباط بین داده‌های اخذ شده و رابط کاربری و نمایش بصری اطلاعات و ماژول‌های دستورات سیستم را خواهد داشت.

۲-۱. مکانیزم چرخدنده

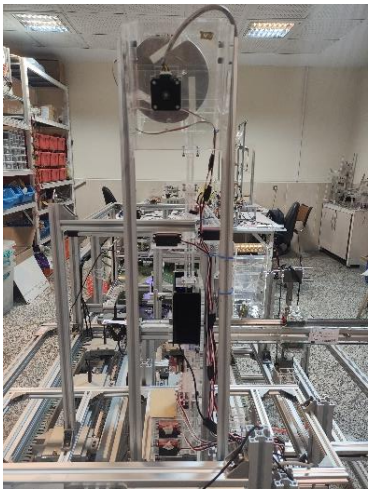
با توجه به بخش‌های متعدد موجود برای مونتاژ قلم انسولین و سطح آمادگی هر کدام از مکانیزم‌های مونتاژ، جهت هوشمندسازی مکانیزم و اعمال رابط کاربری بخش فیدر و سورتر چرخدنده انتخاب شده است.

به طور کلی قلم انسولین شامل دو نوع چرخدنده با نام‌های چرخدنده‌های نوع ۱ و ۲ می‌باشد که به صورت هرزگرد با قطعه‌ی سی سی (سنکرونایزر) در ارتباط‌اند و وظیفه‌ی انتقال قدرت دست را به عهده خواهند داشت. نمونه‌ای از چرخدنده‌های استفاده شده در قلم انسولین به صورت شکل ۴ می‌باشد.

شکل ۴. چرخدنده‌های استفاده شده در قلم انسولین

در این بخش در مرحله‌ی اول، هر دو نوع چرخدنده در مخزنی ذخیره شده اند و توسط یک فیدر مکانیکی چرخان چرخدنده‌ها به سوی خشاب عمودی هدایت می‌شوند. فعالسازی و عملکرد این فیدر توسط موتور DC انجام می‌پذیرد.

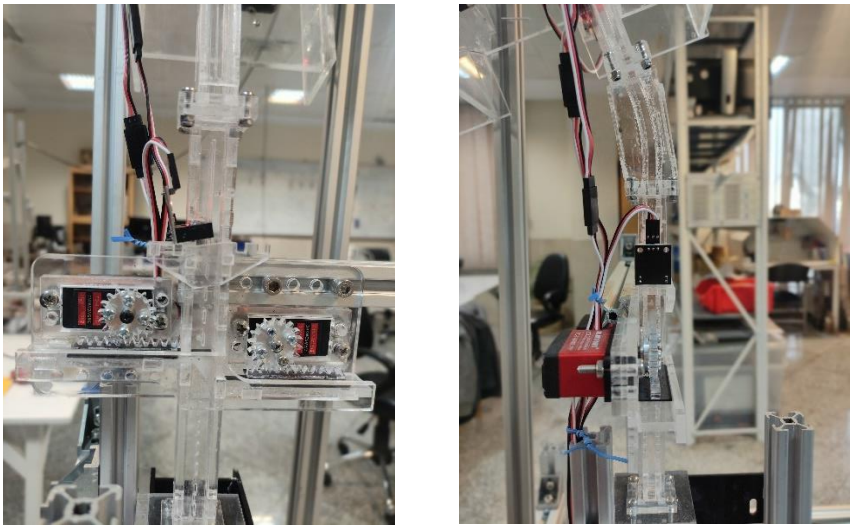
شکل ۵. دو نما از فیدر چرخدنده



شکل ۶. نمایی از فیدر چرخدنده در سیستم کلی مونتاژ

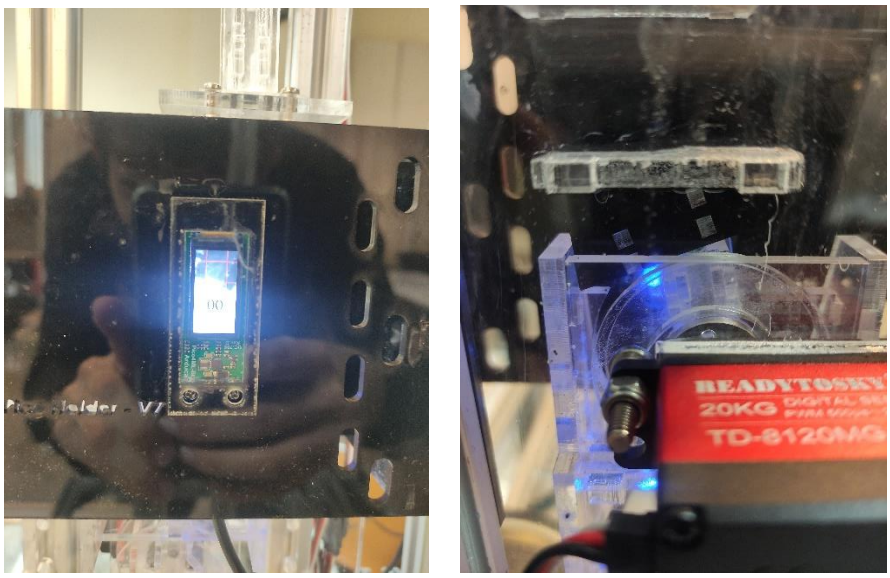
پس از دریافت چرخدنده‌ها توسط فیدر از مخزن، این قطعات در بخشی از خشاب به صورت عمودی ذخیره می‌شوند و توسط یک سنسور تشخیص مانع Opto-Counter مورد پایش قرار می‌گیرند تا در صورت پر

شدن این خشاب از حد تعریف شده، فیدر چرخدنده متوقف گردد. برای عبور دادن چرخدنده‌های ذخیره شده در این بخش به مرحله‌ی بعدی از یک گیت الاکلنگی مبتنی بر سروو موتور استفاده شده است تا بتوانیم در صورت نیاز یک عدد چرخدنده را از این بخش دریافت کرده و به بخش بعدی انتقال دهیم.

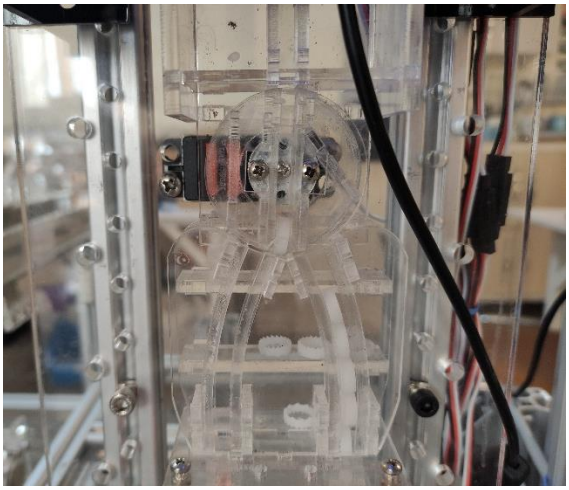


شکل ۷. دو نما از گیت الاکلنگی

در ادامه پس از عبور از این بخش نیز مکانیزم به دنبال تشخیص نوع و جهت چرخدنده‌ی دریافت شده از مرحله‌ی قبلی با استفاده از تکنولوژی بینایی ماشین (ویژن) خواهد بود تا با کمک مکانیزم خشاب چرخان، چرخدنده‌ها در جای خشاب نهایی متناسب با نوع چرخدنده قرار داده شوند. به عبارتی در این بخش چرخدنده‌ها که به صورت تصادفی وارد خشاب شده بودند مرتب و سورت می‌شوند. این فرآیند توسط دو خشاب متحرک انجام می‌پذیرد که پس از تشخیص نوع و جهت چرخدنده‌ها مکانیزم اول بر اساس جهت چرخدنده‌ها را مرتب نموده و در ادامه مکانیزم دوم متناسب با نوع شناسایی شده‌ی چرخدنده، قطعه را در خشاب مناسب می‌اندازد. این دو بخش تحت عنوان گیت‌های فیدر A و B نامگذاری شده‌اند. پس از اتمام فرآیند سورت شدن، چرخدنده‌های نوع یک و دو به صورت جداگانه در خشاب های متناظر و موازی نگهداری می‌شوند.

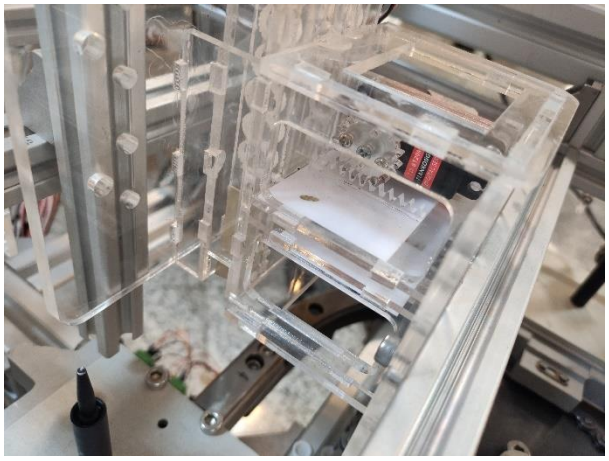


شکل ۸. بخش اول سورتر و ویژن استفاده شده جهت تشخیص نوع و جهت چرخدنده

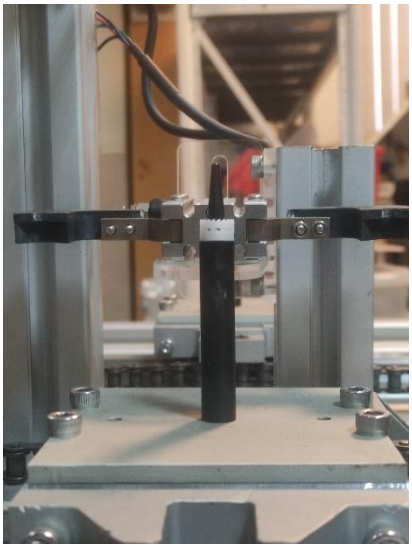


شکل ۹. بخش دوم سورتر چرخدنده

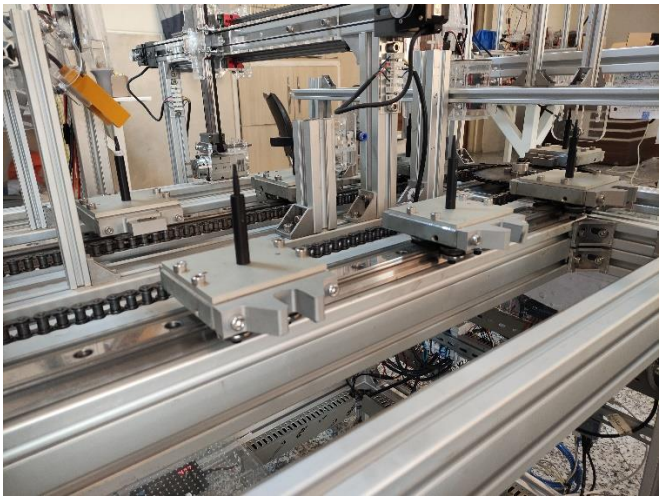
برای انتقال چرخدنده‌ها به فیکسچرهایی که در ادامه بدنه‌ی قلم‌ها به روی آن‌ها و چرخدنده‌ها سوار خواهد شد، یک گیت صفحه‌ای تعبیه شده است تا به صورت تکی از هریک از چرخدنده‌ها را آماده‌ی قرارگیری در فیکسچر نماید و به این ترتیب روند مونتاژ چرخدنده‌ها تکمیل شود. این گیت‌ها نیز توسط سروو موتورها فعال می‌گردند.



شکل ۱۰. گیت نهایی بعد از مرحله‌ی سورتر



شکل ۱۱. نحوه‌ی مطلوب قرارگیری چرخدنده‌ها در فیکسچر



شکل ۱۲. نمایی از واگن‌های حاوی فیکسچر

۲-۲. رابط کاربری

حال رابط کاربری پیشنهادی در قالب این مکانیزم دو وظیفه‌ی اصلی خواهد داشت: دریافت داده‌ها از سنسورها و اعمال دستورات فعالسازی عملگرها. داده‌های سنسورها برای بررسی نحوه‌ی عملکرد سیستم و پایش دائم وضعیت آن بسیار ضروری می‌باشد.

۲-۲-۱. رابط اطلاعاتی

همانطور که در بخش‌های قبلی مطرح شد، در این بخش مسئله‌ی ارتباط بین رابط گرافیکی و مکانیزم مورد نظر می‌باشد. در این بخش بایستی با استفاده از یک پروتوکل ویژه، داده‌های سنسورها و عملگردها از سیستم گرفته شده و به رابط کاربری ارسال گردد و در ادامه فرمان‌ها نیز توسط اپراتور برای رابط گرافیکی تعریف می‌شوند و توسط ارتباط اطلاعاتی به سیستم مونتاژ انتقال می‌یابد.

برای ایجاد این مهم از زبان برنامه‌نویسی پایتون و کتابخانه‌ی Flask استفاده شده است. کتابخانه‌ی Flask یک ابزار و فریم وورکی می‌باشد که با استفاده از آن می‌توان وب پیج‌های ساده و مختصری را ایجاد کرد. این ابزار به زبان ساده می‌تواند با ارسال اشیا درخواستی (Requests) به بخش‌های در ارتباط با گرافیک وب سرور از جمله HTML، CSS و Java Script ارتباط اطلاعاتی را برقرار نماید. بخش ارتباط بین پایتون در قالب دستورات و اطلاعات با استفاده از قالب decorator ها انجام می‌پذیرد. به طوری که طبق آدرسی که در وب سرور تعریف می‌شود، بخش HTML داده‌ها را از پایتون دریافت می‌نماید و یا دستورات تعریف شده در این بخش را به کد پایتون و در ادامه به خود مکانیزم ارسال می‌نماید.

```python
from flask import Flask, jsonify, render_template, request
import requests
import webbrowser
import time
import datetime
import RPi.GPIO as GPIO
from threading import Thread
import os
from dummy_feeder import Dummy_feeder
import sys
from stage1 import *

app = Flask(__name__)

```

شکل ۱۳. بخش اولیه‌ی کد رابط اطلاعاتی در فایل **app.py**

صرفاً با اجرای دستور زیر نیز می‌توانیم یک وب سرور خالی را با استفاده از Flask فعال نماییم.

```python
@app.route('/')
def index():

    return render_template('index.html')

# main driver function
if __name__ == '__main__':

    app.run(debug=True, host='0.0.0.0', port=2000)

```

شکل ۱۴. فرمان ایجاد وب سرور

همچنین یکی از نکات اصلی در آماده‌سازی این بخش، ایجاد ارتباط زمان واقعی بین سیستم اتوماسیون و رابط گرافیکی می‌باشد. این موضوع از این حیث حائز اهمیت است که در یک فرآیند مونتاژ، کاربر و یا اپراتور بایستی اطلاعات و تغییرات موجود در سیستم را بلافاصله دریافت نماید تا بتواند در صورت امکان، تغییرات لازم در سیستم را اعمال نمایند تا روند صحیح عملکرد سیستم حفظ شود. جهت رسیدن به این مهم از کد جاوا اسکریپت استفاده شده‌است. در ابتدا بایستی داده‌هایی که از پایتون به عنوان اطلاعات زمان واقعی نیازمند دریافت هستند را در یک لیست ذخیره‌سازی نماییم و به صورت یک شی JASON تعریف کرده و به بخش جاوا اسکریپت ارسال نماییم. سپس با استفاده از یک تابع تعریف شده در جاوا اسکریپت با نام update_values دیتاهای مورد نظر را که به صورت JASON تعریف شده‌اند را دریافت می‌کنیم. در نهایت تابع تعریف شده را به عنوان ورودی به تابع محلی setInterval می‌دهیم و تعیین می‌نماییم که هر چند میلی ثانیه یکبار رابط کاربری دیتا را از سیستم دریافت نماید.

```
ript src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js"></script>

ript type="text/javascript">

 $SCRIPT_ROOT = {{ request.script_root|tojson|safe }};
 intervalID = setInterval(update_values,1000);

unction update_values() {
    $.getJSON($SCRIPT_ROOT + '/data',

  function(data) {
    $('#result_1').text(data.result[0]);
    $('#result_2').text(data.result[1]);
    $('#result_3').text(data.result[2]);
    $('#result_4').text(data.result[3]);
    $('#result_6').text(data.result[5]);
```

شکل ۱۵. تابع به‌روز رسانی اطلاعات به صورت زمان واقعی

روند کامل راه‌اندازی این بخش به همراه جزئیات در قسمت رابط گرافیکی (Front-End) شرح داده شده است.

۲-۲-۲. رابط گرافیکی (Front-End)

این بخش نمایش ظاهری این رابط کاربری را به عهده خواهد داشت. این قسمت طبق طراحی انجام شده متشکل از چهار جزء خواهد بود. بخش اول جهت دریافت داده‌های مربوط به مکانیزم و وضعیت سنسورها و عملگرها، بخش دوم شامل فرمان‌های تعریف شده برای فعالسازی بخش‌های مختلف مکانیزم، بخش سوم متشکل از یک فلوچارت از مراحل مونتاژ مربوط به این قطعه و بخش چهارم نیز یک جدولی خواهد بود که داده‌های آماری فرآیند مونتاژ را نمایش می‌دهد.

در بخش اول لیست دیتاهای مهمی که از عملکرد دستگاه قابل استخراج است نمایش داده خواهد شد. این بخش با عنوان Dynamic Update در رابط کرافیکی مشخص گردیده است. لیست داده‌هایی که مطلوب می‌باشند به صورت زیر است:

۱- فعال و یا غیر فعال بودن فیدر چرخدنده.
۲- وجود چرخدنده به تعداد کافی در خشاب متصل به فیدر توسط سنسور تشخیص مانع
۳- وضعیت باز یا بسته بودن گیت‌های فیدر چرخان A و B
۴- فعال و یا غیر فعال بودن سورتر چرخدنده
۵- پر بودن خشاب‌های مربوط به چرخدنده‌های ۱ و ۲ پس از سورت شدن
۶- وضعیت باز یا بسته بودن گیت‌های منتهی به سورتر

در بخش کد پایتون بایستی در ابتدا کتابخانه‌ی مربوط به مکانیزم را که قبلاً تهیه شده است را فراخوانی شود. این کتابخانه که صورت فایل پایتون می‌باشد شامل تمام توابع و متغیرهای مربوط به مکانیزم می‌باشد

و با import کردن آن می‌توان به موارد مذکور به طور مستقیم دسترسی داشته باشیم. برای دریافت داده‌های اصلی این بخش از متد Decorator در پایتون استفاده می‌کنیم. به این شکل که با استفاده از این متد یک مسیر با شناسه‌ی data/ را تعریف می‌نماییم. در تابع مربوط به این بخش یک سری گزاره‌های شرطی ساده تعریف می‌شود. به این صورت که مثلاً اگر داده‌ی مربوط به یکی سنورهای تشخیص مانع را در نظر بگیریم، پیامی که این سنور به ما می‌دهد، به صورت مقادیر بولیین True و False خواهد بود. در این حالت با تعریف گزاره‌های شرطی با استفاده از if می‌توان مشخص نمود که سنسور در صورت تولید مقدار True که به معنای دیده شدن شی جلوی سنسور می‌باشد، پیام "Detected" را نمایش دهد و در صورتی که جلوی سنسور مانعی نباشد و چرخدنده‌ای را در خشاب مشاهده ننماید، متغیر مربوط به این سنسور False خواهد بود و در این حالت با شرط تعریف شده بایستی رابط پیام "Not Detected" را گزارش نماید. یا درصورتی که سورتر فعال است وضعیت آن را به صورت Active نمایش دهد و در غیر این صورت پیام Inactive را گزارش نماید. کد این بخش در پایتون به صورت زیر خواهد بود.

```
# DATA TRANSFER ROOT FOR SENSORS & ACTUATORS STATES

@app.route('/data')
def stuff():

    global gear_feeder
    my_feeder = Dummy_feeder(3,3)

    if gear_feeder.state == False :
        res[0] = 'Inactive'
        res[4] = 'red'

    elif gear_feeder.state == True:
        res[0] = 'Active'
        res[4] = '#66cc00'

    if gear_feeder.MagSensor.read() == False:
        res[1] = 'Not Detected'

    elif gear_feeder.MagSensor.read() == True:
        res[1] = 'Detected'

    print(res[1])

    if gear_feeder.GateA.state == 'close':
        res[2] = 'Closed'
        res[5] = 'red'

    elif gear_feeder.GateA.state == 'open':
        res[2] = 'Opened'
        res[5] = '#66cc00'

    if gear_feeder.GateB.state == 'close' :
        res[3] = 'Closed'
        res[12] = 'red'

    elif gear_feeder.GateB.state == 'open':
        res[3] = 'Opened'
        res[12] = '#66cc00'

    if gear_sorter.state == True:
```

شکل ۱۶. تابع تعریف شده جهت دریافت و ارسال داده‌ها در آدرس مدنظر

با آماده شدن دیتاهای مربوط به هر بخش همه‌ی متغیرها را وارد یک متغیر لیست می‌نماییم تا با اعمال فرآیند Jasonify آن‌ها را به Javascript و HTML با استفاده از پروتکل ارتباطی Request ارسال نماییم. پس از دریافت اطلاعات مد نظر این دیتاها را به صورت id هایی در جاوا اسکریپت و بخش تابع به روز رسان تعریف می‌کنیم تا در ادامه بتوانیم با فراخوانی این id ها در اسکریپت، آن‌ها را در HTML نمایش بدهیم. (طبق شکل ۱۴)

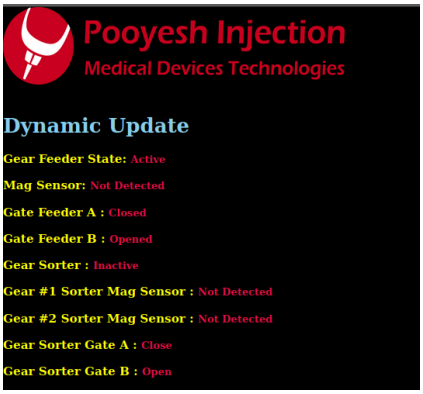
```
    if gear_sorter.gear1MagSensor.read() == True :
        res[7] = 'Detected'

    elif gear_sorter.gear1MagSensor.read() == False:
        res[7] = "Not Detected"

    if gear_sorter.gear2MagSensor.read() == True :
        res[8] = 'Detected'

    elif gear_sorter.gear2MagSensor.read() == False:
        res[8] = "Not Detected"

    if gear_sorter.GateA.state == 'open' :
        res[9] = 'Open'
        res[14] = '#66cc00'

    elif gear_sorter.GateA.state == 'close':
        res[9] = "Close"
        res[14] = 'red'

    if gear_sorter.GateB.state == 'open' :
        res[10] = 'Open'
        res[15] = '#66cc00'

    elif gear_sorter.GateB.state == 'close':
        res[10] = "Close"
        res[15] = 'red'

    if  my_feeder.random_states[2] == 0:
        res[11] = 13

    elif my_feeder.random_states[2] == 1:
        res[11] = 12

    return jsonify(result = res)
```

شکل ۱۷. ادامه‌ی گزاره‌های تابع و برگرداندن داده‌ها به صورت خروجی **Jasonify**

پس از دریافت داده‌ها در بخش جاوا اسکریپت، اسکریپت html را شروع می‌کنیم. در این بخش در ابتدا قالبی که به صورت فایل CSS برای ظاهر کلی و بخشی از اجزای وب سرور در این رابط تعریف نموده‌ایم را فراخوانی می‌کنیم. اجزایی اصلی و ظاهری همچون رنگ، نحوه‌ی قرارگیری اجزای صفحه، فونت متون، جداول، فلوچارت و ... در این بخش تعریف و تعیین شده است. فایل کد CSS به صورت کامل در پیوست‌ها آورده شده است. پس از این بخش، قسمت‌های مد نظر خود را به صورت متنی شروع به تعریف می‌کنیم. به این منظور نیز با ایجاد المان‌های متنی در بخش html متن را در صفحه نمایش می‌دهیم. در ابتدا تیتر مربوط به قسمت اول با نام Dynamic Update که مربوط به نمایش اطلاعات سنسورها و عملگرها می‌باشد را مشخص می‌کنیم. سپس در زیر این تیتر اطلاعات مورد نیاز را گزارش می‌دهیم.

```
<body onload="update_values();">
  <link rel="stylesheet" href='../static/style.css'/><font size = '2'>

  <div class="panel-container">

    <div class="panel-left">
      <img src="../static/Pinjection.png" alt="" width="440" height="100"/>
      <h1 style="color: rgb(135,206,235)" > </h1>
      <hr color="black">
      <h1 style="color: rgb(135,206,235)" > Dynamic Update </h1>

      <p>
        <h3> Gear Feeder State: <font size = '-1'> <span id = result_1 style="color: rgb(228, 13, 67)"> </span> </font> </h3>
        <h3> Mag Sensor: <font size = '-1'> <span id = result_2 style="color: rgb(228, 13, 67)"> </span> </font> </h3>
        <h3> Gate Feeder A : <font size = '-1'> <span id = result_3 style="color: rgb(228, 13, 67)"> </span> </font> </h3>
        <h3> Gate Feeder B :  <font size = '-1'> <span id = result_4 style="color: rgb(228, 13, 67)"> </span> </font> </h3>
        <h3> Gear Sorter :  <font size = '-1'> <span id = result_7 style="color: rgb(228, 13, 67)"> </span> </font> </h3>
        <h3> Gear #1 Sorter Mag Sensor :  <font size = '-1'> <span id = result_8 style="color: rgb(228, 13, 67)"> </span> </font> </h3>
        <h3> Gear #2 Sorter Mag Sensor :  <font size = '-1'> <span id = result_9 style="color: rgb(228, 13, 67)"> </span> </font> </h3>
        <h3> Gear Sorter Gate A :  <font size = '-1'> <span id = result_10 style="color: rgb(228, 13, 67)"> </span> </font> </h3>
        <h3> Gear Sorter Gate B :  <font size = '-1'> <span id = result_11 style="color: rgb(228, 13, 67)"> </span> </font> </h3>
      </p>
```

شکل ۱۸. کد **HTML** مربوط به نمایش داده‌های مد نظر



شکل ۱۹. خروجی مد نظر برای مشاهده‌ی داده‌ها در وب سرور

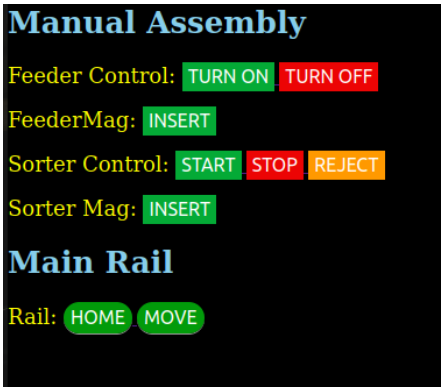
در بخش دوم که مربوط به تعریف دستورات و فرمان‌های مدنظر می‌باشد، باید دستورات مهم و اساسی مکانیزم در فایل‌های مربوط مشخص شود و ارتباط بین دو بخش رابط گرافیکی و مکانیزم برای اجرای این دستورات نیز تعریف گردد. دستورات مد نظر ما برای اعمال در این مکانیزم به صورت زیر می‌باشد:

۱- فعال نمودن مکانیزم فیدر چرخدنده (Feeder Control)
۲- باز و بسته کردن گیت موجود در خشاب (Feeder Mag)
۳- فعالسازی و خارج نمودن چرخدنده‌های نامطلوب از خط مونتاژ (Sorter Control)
۴- باز و بسته نمودن گیت‌های انتهایی سورتر(Sorter Mag)
۵- حرکت مکانیزم ریلی حاوی فیکسچرها (Rail Control)

در این قسمت روند فرآیند برخلاف بخش قبلی از قسمت رابط گرافیکی شروع می‌شود چرا که ورودی مسئله از این بخش گرفته خواهد شد. ابتدا در قسمت کد HTML بخش مربوط به دستورات و نحوه‌ی عملکردی آن‌ها به صورت متنی و بصری تعریف می‌شود. سپس تیترهای مربوط به این بخش را نیز تعریف می‌کنیم که شامل مونتاژ مربوط به مکانیزم چرخدنده‌ها و حرکت مکانیزم ریل و واگن می‌باشد. برای تعریف دستورات از المان <a> استفاده شده‌است. برای ایجاد قابلیت تعریف و اعمال دستورات از طرف کاربر به سیستم از دکمه‌های گرافیکی بهره‌گیری شده که روی هرکدام کارکرد آن‌ها مشخص گردیده‌است. طبق کد نیز برای هر یک از این عملگرها یک مسیر آدرس مشخص و یکتایی در وب سرور تعیین شده که در صورت کلیک نمودن هر دکمه، وب سرور به مسیر تعیین شده هدایت می‌شود و این آدرس به بخش مد نظر در فایل پایتون ارسال می‌شود. شکل ظاهری این دکمه‌ها نیز تحت عنوان button class در کد مربوط به CSS تدوین شده‌است.

```
<h1 style="color: rgb(135,206,235)">  Manual Assembly</h1>

<p> <font size = '4'> Feeder Control: </font>
  <a href="/on"> <button class="button-on" > TURN ON </button> </a>
  <a href="/off"> <button class="button-off"> TURN OFF </button> </a>
</p>


<p> <font size = '4'> FeederMag: </font>
  <a href="/rotate"> <button button class="button-on"> INSERT </button> </a>
</p>

<p> <font size = '4'> Sorter Control: </font>
  <a href="/sorterStart"> <button button class="button-on"> START </button> </a>
  <a href="/sorterStop"> <button button class="button-off"> STOP </button> </a>
  <a href="/reject"> <button button class="button-reject"> REJECT </button> </a>
</p>

<p> <font size = '4'> Sorter Mag: </font>
  <a href="/magInsert"> <button button class="button-on"> INSERT </button> </a>
</p>

<h1 style="color: rgb(135,206,235)"> Main Rail </h1>

<p> <font size = '4'> Rail:  </font>
  <a href="/mainRailIn"> <button class="button">HOME</button> </a>
  <a href="/mainRailMove"> <button class="button">MOVE</button> </a>
</p>

<div>
```

شکل ۲۰. کد **HTML** جهت ارسال فرمان‌های مد نظر به مکانیزم‌ها



شکل ۲۱. خروجی مد نظر برای ارسال فرمان‌ها در مکانیزم در وب سرور

پس از این بخش در رابط گرافیکی، بایستی متناظراً کدهای مربوط به رابط اطلاعاتی نیز تعریف گردد. در وهله‌ی اول دستورات مربوط به فرمان‌های مکانیزم را که در کتابخانه‌ی از پیش تهیه شده، تعریف شده است را فراخوانی می‌کنیم. با اینکار مجدداً با استفاده از متد دکوریتور(Decorator)، تابعی را با نام action در فایل پایتون تعریف می‌نماییم که به عنوان ورودی نوع فرمان را از کاربر دریافت می‌کند. این تابع صرفاً از یکسری گزاره‌های شرطی تشکیل شده است که با دریافت دستور از کابر مطابقاً فرمان مطلوب را در مکانیزم اجرا می‌کند.

```
#  TRANSFER ROOT FOR THE ACTIONS & COMMANDS OF THE DECVICE

@app.route("/<action>")
def action(action):

    global gear_feeder
    global gear_sorter
    global mail_rail

    if action == "on":
        gear_feeder.start()
    elif action == "off":
        gear_feeder.stop()

    if action == 'rotate':
        gear_feeder.MagInsert()

    if action == 'magInsert':
        gear_sorter.MagInsert()

    if action == 'sorterStart':
        gear_sorter.start()

    elif action == 'sorterStop':
        gear_sorter.stop()

    if action == 'reject':
        gear_sorter.reject()

    if action == 'mainRailIn':
        main_rail.home()

    elif action == 'mainRailMove':
        main_rail.move()

    return render_template('index.html')

```

شکل ۲۲. تابع تعریف شده در رابط ارتباطی جهت دریافت دستورات مکانیزم

در بخش سوم شامل فلوچارتی از فرآیند فید و سورت شدن چرخدنده‌ها و آماده شدن آن‌ها برای مراحل بعدی مونتاژ می‌باشد. این نمودار به صورتی می‌باشد که هر کدام از مراحل این فرآیند را نمایش می‌دهد و به نوعی اطلاعات موجود در بخش Dyanmic Update را به صورت بصری و گرافیکی بیان می‌کند که برای داشتن دیدی کلی و آسان‌تر از نحوه‌ی عملکرد مکانیزم مفید است. این بخش نیز با استفاده از متغیرهای مربوط به نحوه‌ی عملکرد مکانیزم‌ها، وضعیت سنسورها و عملگرها را نمایش می‌دهد. به طور جزئی‌تر در صورتی که عملگرهای مدنظر بنا به الگوریتم مکانیزم یا فرمان اپراتور به سیستم فعال شوند و شروع به کار نمایند، در این صورت شماتیک مربوط به هر کدام از این فرآیندها به رنگ سبز در می‌آید و در صورت عدم فعال شدن و یا عملکرد ناقص، شماتیک به رنگ قرمز در می‌آید. مراحل این بخش به این ترتیب خواهد بود:

۱- وضعیت عملکرد فیدر (Feeder Sorter: Step 1)

۲- باز بودن گیت A فیدر (Gate Feeder A: Step 2)

۳- باز بودن گیت B فیدر(Gate Feeder B: Step 3)

۴- باز بودن گیت A سورتر (Sorter: Step 4)

۵- باز بودن گیت B سورتر (Gate Sorter A: Step 5)

۶- وضعیت عملکرد سورتر (Gate Sorter B: Step 6)

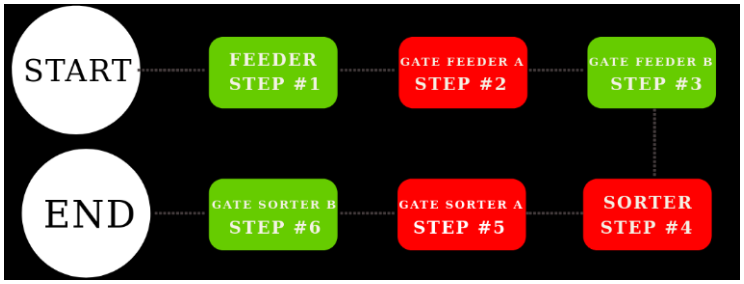
در بخش رابط اطلاعاتی این بخش نیز مشابه بخش اول که جهت دریافت داده‌های مربوط به مکانیزم و وضعیت سنسورها و عملگرها می‌بود، با استفاده از یکسری تعاریف گزاره‌های شرطی نوع رنگ هر بلوک شماتیکی تعیین می‌شود (شکل۲۱) .

تنظیمات ظاهری این نمودار فرآیندی به صورت Container در کد مربوط CSS تعریف شده است. قسمت مربوط HTML نیز به شکل زیر خواهد بود. لازم به ذکر است که در شکل زیر به خاطر طولانی بودن کد، صرفاً بخش محدودی از کد فلوچارت نمایش داده شده است و نسخه‌ی کامل این قسمت در بخش پیوست‌ها قابل مشاهده است.

```
<article class="panel-right">

<div class="wrapper">
  <div class="svg-container">
    <svg version="1.1" viewBox="-10 0 520 900" preserveAspectRatio="xMinYMin meet" style ='float' class="svg-content">

      <defs>
        <marker id="arrow" markerWidth="4" markerHeight="10" viewBox="-2 -4 4 4" refX="0" refY="0" markerUnits="strokeWidth" orient=
        | <polyline points="2,-2 0,0 2,2" stroke="#443c3d" stroke-width="0.75px" fill="none"/>
        </marker>
      </defs>

      <g class="box-group">
        <g transform="translate(-10)">

          <circle fill="white" cx="55" cy="50" r="45" opacity="1" />
          <text x="18" y="58" font-family="Open Sans Condensed" font-size="21" stroke="none" fill="#000" font-weight="100" style="te
          <line x1="98" x2="135" y1="50" y2="50" stroke-width="2" stroke="#443c3d" stroke-dasharray="2,1" />
        </g>

        <line x1="100" x2="136" y1="50" y2="50" stroke-width="2" stroke="#443c3d" stroke-dasharray="2,1" />
        <g transform="translate(136)">

        <rect id='box1'/>
        <text x="16" y="47" font-family="Open Sans Condensed" font-size="12" stroke="none" fill="#f5f3e7" font-weight="900" style="t
        | <tspan x="16" dy="17">Step #1</tspan>
        </text>
        </g>
```

شکل ۲۳. بخش مربوط به فلوچارت در کد **HTML**

نمایش ظاهری این فلو چارت در نهایت به صورت زیر خواهد بود. لازم به ذکر است که برای آشکار شدن اولویت توالی مراحل مونتاژ، دو شماتیک استاتیک Start و Stop به عنوان نقاط شروع و پایانی فرآیند نمایش داده شده است.



شکل ۲۴. خروجی فلوچارت فرآیند مونتاژ در وب سرور

در نهایت بخش چهارم و آخر این رابط گرافیکی قسمت مربوط به داده‌های آماری مربوط به عملکرد سیستم خواهد بود که نمایش دهد که در تکرارهای بالا نحوه‌ی عملکرد مکانیزم مونتاژ به چه شکل خواهد بود. هدف این قسمت نمایش آماری از قبیل:

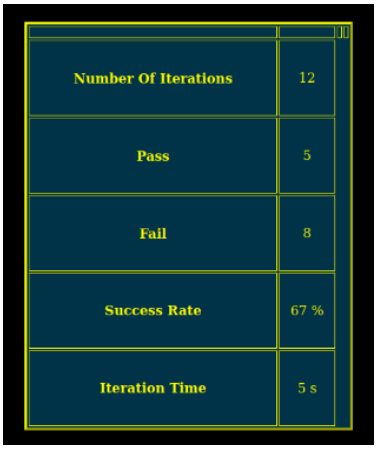
۱- تعداد مراحل طی شده
۲- تعداد مونتاژهای صحیح
۳- تعداد مونتاژهای غلط
۴- درصد موفقیت مونتاژ
۵- مدت زمان انجام مونتاژ

کد قسمت مربوط به طراحی جدول نیز به شکل زیر خواهد بود. لازم به ذکر است که همچون دیگر المان‌های موجود، جهت تنظیم ظاهر جدول مد نظر، یک کلاس متناظر در بخش css تعریف شده و جزئیات مطلوب در داخل آن کلاس مشخص گردیده است.

```
<div class="flex-container" >
  <div class="flex-up">
  <div class="wrp">
<table border="3">
 <tr>
    <th> </th>
    <th> </th>
    <th> </th>
    <th> </th>
 </tr>

 <tr>
    <th> Number Of Iterations </th>
    <td> <span id = result_12 style="color: yellow"> </span> </td>
 </tr>

 <tr>
    <th> Pass </th>
    <td> 5 </td>
 </tr>

 <tr>
    <th>Fail</th>
    <td> 8 </td>
 </tr>

 <tr>
    <th>Success Rate</th>
    <td> 67 % </td>
 </tr>

 <tr>
 <th> Iteration Time </th>
 <td> 5 s </td>
 </tr>

 </table>
</div>
</nav>
</div>
</div>
```

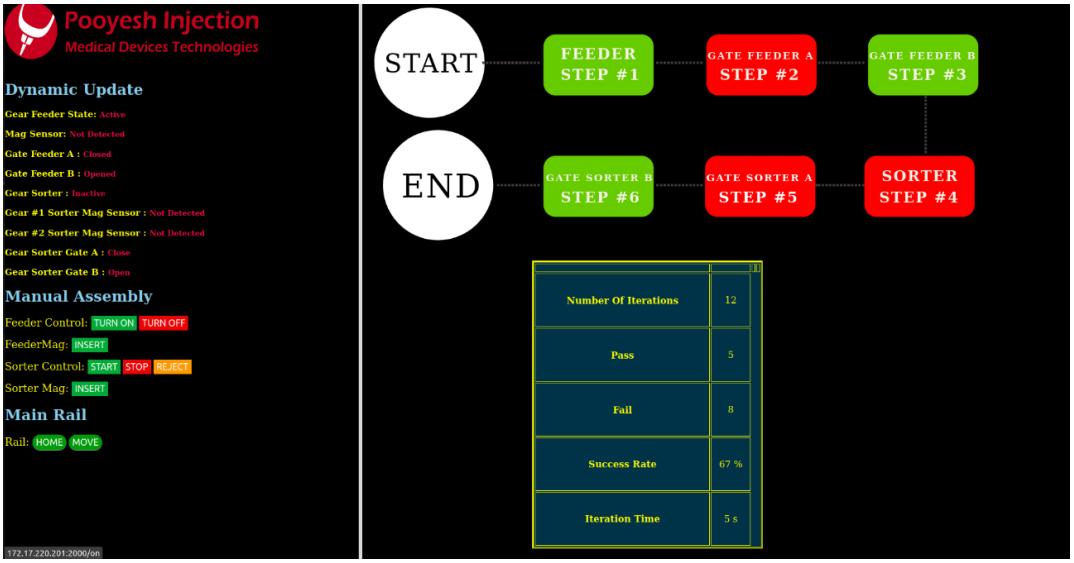
شکل ۲۵. بخش مربوط به طراحی جدول آماری در کد **HTML**

در نهایت نمایش نهایی این جدول در وب سرور نیز به این صورت خواهد بود.



شکل ۲۶. خروجی جدول آماری مونتاژ در وب سرور

با کنار هم قرار دادن این بخش‌ها در صفحه نمایش، در نهایت ظاهر نهایی وب سرور مربوط به رابط گرافیکی سیستم مونتاژ اتوماتیک چرخدنده‌ها به صورت زیر خواهد بود.



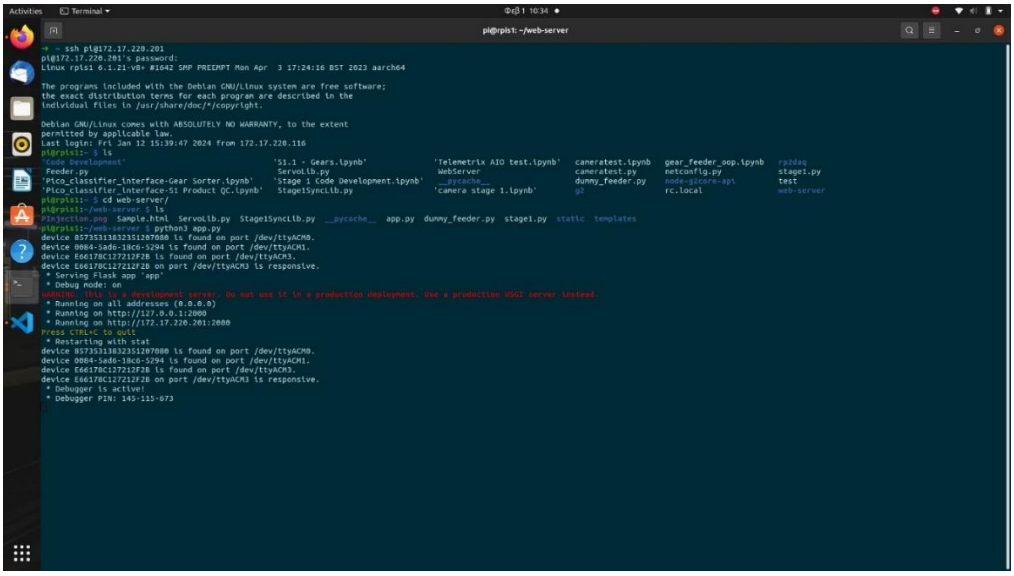
شکل ۲۷. نمایی نهایی از رابط کاربری مکانیزم مونتاژ چرخدنده‌های قلم انسولین

نحوه‌ی اجرای رابط گرافیکی

همانطور که در مراحل قبلی تشریح داده شده، رابط کاربری تدوین شده برای سیستم مونتاژ قلم انسولین، یک وب سروری می‌باشد که به اصطلاح در localhost میکروکامپیوترهای موسوم به رزپبری پای اجرا می‌شود. ولی علارغم این موضوع، توسط دیگر کامپیوترها و دستگاه‌های شخصی که به اینترنت مشترک با بوردهای رزپبری پای متصل هستند، قابل دسترسی می‌باشد که شیوه‌ی این دسترسی در ادامه شرح داده خواهد شد.

۳-۱. فعالسازی اولیه‌ی وب سرور

جهت ایجاد این وب سرور در ابتدا بایستی کد مربوط به رابط اطلاعاتی پایتون با نام app.py در داخل رزپبری پای اجرا شود. برای این منظور دو روش را می‌توان در پیش گرفت. روش اول و آسان‌تر این خواهد بود که با استفاده از یک مانیتور به مینی کامپیوتر رزپبری پای وصل شد و به صورت مستقیم به داخل سیستم آن دسترسی داشته باشیم. در صورت عدم وجود این امکان، می‌توان از کامپیوتر شخصی و پروتکل ssh که مبتنی بر شبکه‌های بی‌سیم است به داخل سیستم رزپبری پای متصل شد و ادامه‌ی روند را طی نمود. در طول طراحی و ساخت این وب سرور و تست عملکرد آن با توجه به محدودیت‌های موجود، از روش دوم برای دسترسی به سیستم عامل رزپبری پای استفاده شده است. سیستم عامل استفاده شده در کامپیوتر شخصی جهت انجام اینکار ورژن Uubntu از سری سیستم‌عامل‌های Linux می‌باشد. توالی مراحل اجرای فایل پایتون اینگونه خواهد بود که ابتدا ترمینال لینوکس را بالا می‌آوریم. بعد از انجام آن با استفاده از hostname و username رزپبری پای مربوط، دستور ssh را اجرا می‌کنیم تا به آن متصل شویم. بدر ادامه به پوشه‌ی مربوط به کدها و اطلاعات حاوی این وب سرور می‌رویم و در نهایت کد پایتون مذکور را اجرا می‌کنیم. تمامی مراحل مشخص شده در ترمینال به صورت زیر خواهد بود.



شکل ۲۸. دستورات لازم برای راه‌اندازی وب سرور در ترمینال لینوکس

۳-۱. اجرای وب سرور

همانطور که مطرح شد پس از ران نمودن این بخش در ترمینال، وب سرور فعال خواهد شد و می‌توان به آن دستری پیدا نمود. برای اینکار نیز طبق IP رزپبری پای که به آن وصل شده‌ایم و پورتی که در بخش پایتون برای این وب سرور تعریف نموده‌ایم می‌توانیم به وب سرور دسترسی داشته باشیم. به این صورت که با وارد کردن آدرس IP address:port در مرورگر صفحه‌ی مربوط به وب سرور به نمایش داده خواهد شد. در شکل (۲۸) ملاحظه می‌شود که در پیام‌های خروجی اجرای فایل پایتون نیز پیام مربوط به آدرسی که وب سرور در آن در حال اجرا می‌باشد آورده شده است که با کلیک بر روی آن می‌توانیم رابط کاربری ایجاد شده را باز نماییم.

نتیجه‌گیری و پیشنهادات

در بخش‌های قبلی ملاحظه نمودیم که با استفاده از Flask توانستیم یک ارتباط بین کاربر و دستگاه‌های مونتاژ ایجاد نماییم. این رابط این امکان را به ما می‌دهد تا بتوانیم وضعیت دستگاه را در هر لحظه بسنجیم و فرمان‌های مطلوب جهت اجرا توسط مکانیزم‌ها را نیز ارسال نماییم. در نگاه کلی، این روش نسبت به روش‌های دیگر دارای این مزیت می‌باشد که می‌توان مطابق با هر دستگاه به آسانی روند شخصی‌سازی (Customizing) این رابط را انجام داد. با توجه به اینکه هدف در آینده‌ی بلند مدت، استفاده از این رابط کاربری برای بخش‌ها و دستگاه‌های مختلف مونتاژ می‌باشد، بایستی رابط طراحی شده دارای قابلیت پیاده‌سازی بر روی دیگر دستگاه‌ها بدون نیاز به تغییرات و هزینه‌ی زیاد باشد. از این حیث روش اتخاذ شده نسبت به دیگر راه‌حل‌ها برتر است. با این حال وجود این قابلیت، روند کار با این روش را تا حدودی سخت و زمانبر می‌کند. در این روش تقریباً تمام المان‌ها توسط طراح تعریف و ایجاد شده است. راه‌حل‌های دیگری همچون SCADA:Node-RED قالبی آماده جهت انجام پروژه‌های طراحی رابط‌های انسان و ماشین می‌باشند که بسیاری از المان‌های گرافیکی مطلوب در آن به صورت آماده قابل استفاده می‌شود. روند پیاده‌سازی این روش نیز از لحاظ زمانی کم بوده و دارای مراحل کمتری می‌باشد.

به عنوان نمایی از آینده‌ی پروژه، می‌توان از قالب‌های آماده جهت ارتباط اطلاعاتی استفاده کرد و برای اینکه بتوانیم روند شخصی‌سازی را نیز پیاده نماییم، مجدد مشابه روند طی شده در طراحی این پروژه از کدهای HTML و CSS می‌توان بهره برد تا بتوانیم رابط ماشین و انسان مطلوب و بهینه را با در نظر داشتن مزیت‌های راه‌حل‌های متعدد طراحی نماییم.

مراجع

۱ – History of HMI, 2022 August, Tehran International Complex

۲- Research Gate, Programming of HMI: Actuation State of The Servomotors

پیوست‌ها

۴-۱. کد مربوط به Back-End رابط کاربری با نام app.py

app.py

```
from flask import Flask, jsonify, render_template, request
import requests
import webbrowser
import time
import datetime
import RPi.GPIO as GPIO
from threading import Thread
import os
from dummy_feeder import Dummy_feeder
import sys
from stage1 import *

app = Flask(__name__)

res = [None] * 20

# OBJECT DEFINITION

gear_feeder = Gear_Feeder()
gear_sorter = Gear_Sorter(gear_feeder)
main_rail = Main_Rail()


# DATA TRANSFER ROOT FOR SENSORS & ACTUATORS STATES

@app.route('/data')
def stuff():

    global gear_feeder
    my_feeder = Dummy_feeder(3,3)

    if gear_feeder.state == False :
        res[0] = 'Inactive'
        res[4] = 'red'

    elif gear_feeder.state == True:
        res[0] = 'Active'
        res[4] = '#66cc00'

    if gear_feeder.MagSensor.read() == False:
        res[1] = 'Not Detected'

    elif gear_feeder.MagSensor.read() == True:
        res[1] = 'Detected'
```

```
    if gear_feeder.GateA.state == 'close':
        res[2] = 'Closed'
        res[5] = 'red'

    elif gear_feeder.GateA.state == 'open':
        res[2] = 'Opened'
        res[5] = '#66cc00'

    if gear_feeder.GateB.state == 'close' :
        res[3] = 'Closed'
        res[12] = 'red'

    elif gear_feeder.GateB.state == 'open':
        res[3] = 'Opened'
        res[12] = '#66cc00'

    if gear_sorter.state == True:
        res[6] = 'Active'
        res[13] = '#66cc00'

    elif gear_sorter.state == False:
        res[6] = "Inactive"
        res[13] = 'red'

    if gear_sorter.gear1MagSensor.read() == True :
        res[7] = 'Detected'

    elif gear_sorter.gear1MagSensor.read() == False:
        res[7] = "Not Detected"

    if gear_sorter.gear2MagSensor.read() == True :
        res[8] = 'Detected'

    elif gear_sorter.gear2MagSensor.read() == False:
        res[8] = "Not Detected"

    if gear_sorter.GateA.state == 'open' :
        res[9] = 'Open'
        res[14] = '#66cc00'

    elif gear_sorter.GateA.state == 'close':
        res[9] = "Close"
        res[14] = 'red'

    if gear_sorter.GateB.state == 'open' :
        res[10] = 'Open'
        res[15] = '#66cc00'

    elif gear_sorter.GateB.state == 'close':
        res[10] = "Close"
        res[15] = 'red'

    if  my_feeder.random_states[2] == 0:
        res[11] = 13

    elif my_feeder.random_states[2] == 1:
        res[11] = 12

    return jsonify(result = res)
```

```
#  TRANSFER ROOT FOR THE ACTIONS & COMMANDS OF THE DECVICE

@app.route("/<action>")
def action(action):

    global gear_feeder
    global gear_sorter
    global mail_rail

    if action == "on":
        gear_feeder.start()
    elif action == "off":
        gear_feeder.stop()

    if action == 'rotate':
        gear_feeder.MagInsert()

    if action == 'magInsert':
        gear_sorter.MagInsert()

    if action == 'sorterStart':
        gear_sorter.start()

    elif action == 'sorterStop':
        gear_sorter.stop()

    if action == 'reject':
        gear_sorter.reject()

    if action == 'mainRailIn':
        main_rail.home()

    elif action == 'mainRailMove':
        main_rail.move()

    return render_template('index.html')

@app.route('/')
def index():

    return render_template('index.html')

# main driver function
if __name__ == '__main__':

    app.run(debug=True, host='0.0.0.0', port=2000)
```

۴-۲. کد اول مربوط به Front-End رابط کاربری با نام style.css

style.css

```
html,body {
    height: 100%;
    overflow: hidden;
    padding: 0;
    margin: 0;
    background: black;
    color: yellow;
}



.wrapper {
    padding: 0px 0px;
  }

  .svg-container {
      display: inline-block;
      position: relative;
      width: 90%;
      padding-bottom: 35%;
      vertical-align: middle;
      overflow: hidden;
    background: #000000;
  }

  .svg-content {
      display: inline-block;
      position: absolute;
      top: 0;
      left: 0;
  }


  .wrp {
    padding: 20px 295px; /* Outer Padding */
    }

table {
    padding: 0px 0px;  /* Inner Padding */
    vertical-align: right;
    color: yellow;
    background-color: #003347;
    height: 500px;
    width: 400px;
    border: 10;
 }

 td {
    text-align: center; /* second column*/
}

th {
    text-align: center; /* first column*/
}
```

```css
.panel-container {
    display: flex;
    flex-direction: row;

    justify-content: space-around;
    flex-wrap: nowrap;
    align-items: stretch;
}


.panel-left {
    flex: 2;  /* manual resize */
    width: 3000px;
    color: "red"
}

.splitter {
    flex: 0; /* manual resize */
    width: 1200px;
    border: 3px solid rgba(245, 245, 245, 0.884);
}

.panel-right {
    flex: 4;
    width: 2000px;
    justify-content: center;
    align-items: center;
}

.flex-container {
    display: flex;
    flex-direction: column;
    height: 100%;
    overflow: auto;
}

.felx-up {
    flex: 100;  /* manual resize */
    width: 300px;
}
.splitter-vertical {
    flex: 10; /* manual resize */
    width: 18px;
}
.flex-down {
    flex: 1; /* resizable */
}

/* Use Flexbox for the row */

.button-on {
    background-color: #04aa36; /* Green */
    border: none;
    color: white;
    padding: 3px 5px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
  }
```

```css
.button-off {
    background-color: #ec0404; /* Green */
    border: none;
    color: white;
    padding: 3px 5px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
}

.button-reject {
    background-color: #ff9900; /* Green */
    border: none;
    color: white;
    padding: 3px 5px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
}


.button {
    display: inline-block;
    padding: 4px 6px;
    font-size: 24px;
    cursor: pointer;
    text-align: center;
    text-decoration: none;
    outline: none;
    color: #fff;
    background-color: #029b0a;
    border: none;
    border-radius: 15px;
    box-shadow: 0 1px #999;
    font-size: 16px;
  }

  .button:hover {background-color: #02b60b}

  .button:active {
    background-color: #3e8e41;
    box-shadow: 0 5px #666;
    transform: translateY(4px);

  }
```

۴-۲. کد دوم مربوط به Front-End رابط کاربری با نام index.html

index.html

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js"></script>
<script type="text/javascript">
  var $SCRIPT_ROOT = {
    {
      request.script_root | tojson | safe
    }
  };
  var intervalID = setInterval(update_values, 1000);

  function update_values() {
    $.getJSON($SCRIPT_ROOT + '/data', function(data) {
      $('#result_1').text(data.result[0]);
      $('#result_2').text(data.result[1]);
      $('#result_3').text(data.result[2]);
      $('#result_4').text(data.result[3]);
      $('#result_6').text(data.result[5]);
      $('#result_7').text(data.result[6]);
      $('#result_8').text(data.result[7]);
      $('#result_9').text(data.result[8]);
      $('#result_10').text(data.result[9]);
      $('#result_11').text(data.result[10]);
      $('#result_12').text(data.result[11]);
      $('#result_13').text(data.result[12]);
      const box_color_1 = data.result[4];
      var box1 = document.getElementById("box1");
      box1.style.width = "90";
      box1.style.height = "50";
      box1.style.fill = box_color_1;
      const box_color_2 = data.result[5];
      var box2 = document.getElementById("box2");
      box2.style.width = "90";
      box2.style.height = "50";
      box2.style.fill = box_color_2;
      const box_color_3 = data.result[12];
      var box3 = document.getElementById("box3");
      box3.style.width = "90";
      box3.style.height = "50";
      box3.style.fill = box_color_3;
      const box_color_4 = data.result[13];
      var box4 = document.getElementById("box4");
      box4.style.width = "90";
      box4.style.height = "50";
      box4.style.fill = box_color_4;
      const box_color_5 = data.result[14];
      var box5 = document.getElementById("box5");
      box5.style.height = "50";
      box5.style.width = "90";
      box5.style.fill = box_color_5;
      const box_color_6 = data.result[15];
      var box6 = document.getElementById("box6");
      box6.style.width = "90";
      box6.style.height = "50";
      box6.style.fill = box_color_6;
      console.log(data.result);
    });
```

```
    box1.style.right = "40px";
    box1.setAttribute("y", 26.8);
    box1.setAttribute("x", 2);
    box1.setAttribute("rx", 10);
    box1.setAttribute("ry", 10);
    box2.style.right = "40px";
    box2.setAttribute("y", 26.8);
    box2.setAttribute("x", 3);
    box2.setAttribute("rx", 10);
    box2.setAttribute("ry", 10);
    box3.style.right = "40px";
    box3.setAttribute("y", 26.8);
    box3.setAttribute("x", 3);
    box3.setAttribute("rx", 10);
    box3.setAttribute("ry", 10);
    box4.style.right = "40px";
    box4.setAttribute("y", 26);
    box4.setAttribute("x", 0);
    box4.setAttribute("rx", 10);
    box4.setAttribute("ry", 10);
    box5.style.right = "40px";
    box5.setAttribute("y", 26);
    box5.setAttribute("x", 5);
    box5.setAttribute("rx", 10);
    box5.setAttribute("ry", 10);
    box6.style.right = "40px";
    box6.setAttribute("y", 26);
    box6.setAttribute("x", 2);
    box6.setAttribute("rx", 10);
    box6.setAttribute("ry", 10);
  };

  function stopTextColor() {
    clearInterval(intervalID);
  }
</script>
<script>
  var svg = document.getElementsByTagName("svg"),
    rect = document.getElementsByTagName("rect"),
    text = document.createElementNS("http://www.w3.org/2000/svg", "text");
  text.setAttribute("x", "10"), text.setAttribute("y", "20"), text.setAttribute("fill",
"#000"), text.textContent = ""
</script>
<body onload="update_values()">
  <link rel="stylesheet" href="../static/style.css">
  <font size="2">
    <div class="panel-container">
      <div class="panel-left">
        <img src="../static/Pinjection.png" alt="" width="440" height="100">
        <h1 style="color:#87ceeb"></h1>
        <hr color="black">
        <h1 style="color:#87ceeb">Dynamic Update</h1>
        <p>
        <h3>Gear Feeder State: <font size="-1">
            <span id="result_1" style="color:#e40d43"></span>
          </font>
        </h3>
        <h3>Mag Sensor: <font size="-1">
            <span id="result_2" style="color:#e40d43"></span>
          </font>
```

```
</h3>
<h3>Gate Feeder A : <font size="-1">
    <span id="result_3" style="color:#e40d43"></span>
  </font>
</h3>
<h3>Gate Feeder B : <font size="-1">
    <span id="result_4" style="color:#e40d43"></span>
  </font>
</h3>
<h3>Gear Sorter : <font size="-1">
    <span id="result_7" style="color:#e40d43"></span>
  </font>
</h3>
<h3>Gear #1 Sorter Mag Sensor : <font size="-1">
    <span id="result_8" style="color:#e40d43"></span>
  </font>
</h3>
<h3>Gear #2 Sorter Mag Sensor : <font size="-1">
    <span id="result_9" style="color:#e40d43"></span>
  </font>
</h3>
<h3>Gear Sorter Gate A : <font size="-1">
    <span id="result_10" style="color:#e40d43"></span>
  </font>
</h3>
<h3>Gear Sorter Gate B : <font size="-1">
    <span id="result_11" style="color:#e40d43"></span>
  </font>
</h3>
</p>
<h1 style="color:#87ceeb">Manual Assembly</h1>
<p>
  <font size="4">Feeder Control:</font>
  <a href="/on">
    <button class="button-on">TURN ON</button>
  </a>
  <a href="/off">
    <button class="button-off">TURN OFF</button>
  </a>
</p>
<p>
  <font size="4">FeederMag:</font>
  <a href="/rotate">
    <button button class="button-on">INSERT</button>
  </a>
</p>
<p>
  <font size="4">Sorter Control:</font>
  <a href="/sorterStart">
    <button button class="button-on">START</button>
  </a>
  <a href="/sorterStop">
    <button button class="button-off">STOP</button>
  </a>
  <a href="/reject">
    <button button class="button-reject">REJECT</button>
  </a>
</p>
<p>
  <font size="4">Sorter Mag:</font>
  <a href="/magInsert">
```

```
            <button button class="button-on">INSERT</button>
          </a>
        </p>
        <h1 style="color:#87ceeb">Main Rail</h1>
        <p>
          <font size="4">Rail:</font>
          <a href="/mainRailIn">
            <button class="button">HOME</button>
          </a>
          <a href="/mainRailMove">
            <button class="button">MOVE</button>
          </a>
        </p>
        <div>
          <div class="splitter-vertical"></div>
        </div>
      </div>
      <div class="splitter"></div>
      <article class="panel-right">
        <div class="wrapper">
          <div class="svg-container">
            <svg version="1.1" viewBox="-10 0 520 900" preserveAspectRatio="xMinYMin
meet" style="" class="svg-content">
              <defs>
                <marker id="arrow" markerWidth="4" markerHeight="10" viewBox="-2 -4 4
4" refX="0" refY="0" markerUnits="strokeWidth" orient="auto">
                  <polyline points="2,-2 0,0 2,2" stroke="#443c3d" stroke-
width="0.75px" fill="none" />
                </marker>
              </defs>
              <g class="box-group">
                <g transform="translate(-10)">
                  <circle fill="white" cx="55" cy="50" r="45" opacity="1" />
                  <text x="18" y="58" font-family="Open Sans Condensed" font-size="21"
stroke="none" fill="#000" font-weight="100" style="text-transform:uppercase;letter-
spacing:1px">Start</text>
                  <line x1="98" x2="135" y1="50" y2="50" stroke-width="2"
stroke="#443c3d" stroke-dasharray="2,1" />
                </g>
                <line x1="100" x2="136" y1="50" y2="50" stroke-width="2"
stroke="#443c3d" stroke-dasharray="2,1" />
                <g transform="translate(136)">
                  <rect id="box1" />
                  <text x="16" y="47" font-family="Open Sans Condensed" font-size="12"
stroke="none" fill="#f5f3e7" font-weight="900" style="text-transform:uppercase;letter-
spacing:1px">Feeder <tspan x="16" dy="17">Step #1</tspan>
                  </text>
                </g>
                <line x1="230" x2="268" y1="50" y2="50" stroke-width="2"
stroke="#443c3d" stroke-dasharray="2,1" />
                <g transform="translate(268)">
                  <rect id="box2" />
                  <text x="4" y="47" font-family="Open Sans Condensed" font-size="8"
stroke="none" fill="#f5f3e7" font-weight="900" style="text-transform:uppercase;letter-
spacing:1px">Gate Feeder A <tspan x="14" dy="17" color="red" font-size="12">Step
#2</tspan>
                  </text>
                </g>
                <line x1="362" x2="400" y1="50" y2="50" stroke-width="2"
```

```
stroke="#443c3d" stroke-dasharray="2,1" />
                <g transform="translate(400)">
                  <rect id="box3" />
                  <text x="4" y="47" font-family="Open Sans Condensed" font-size="8"
stroke="none" fill="#f5f3e7" font-weight="1000" style="text-transform:uppercase;letter-
spacing:1px">Gate Feeder B <tspan id="my-tspan" font-size="12" x="19" dy="17">Step
#3</tspan>
                  </text>
                </g>
              </g>
              <line x1="450" x2="450" y1="77" y2="124" stroke-width="2"
stroke="#443c3d" stroke-dasharray="2,1" />
              <g class="box-group" transform="translate(0,100)">
                <line x1="100" x2="136" y1="50" y2="50" stroke-width="2"
stroke="#443c3d" stroke-dasharray="2,1" />
                <g transform="translate(136)">
                  <rect id="box6" />
                  <text x="4" y="47" font-family="Open Sans Condensed" font-size="8"
stroke="none" fill="#f5f3e7" font-weight="900" style="text-transform:uppercase;letter-
spacing:1px">Gate Sorter B <tspan x="16" dy="17" font-size="12">Step #6</tspan>
                  </text>
                </g>
                <line x1="230" x2="268" y1="50" y2="50" stroke-width="2"
stroke="#443c3d" stroke-dasharray="2,1" />
                <g transform="translate(265)">
                  <rect id="box5" x="2" y="25" />
                  <text x="6" y="47" font-family="Open Sans Condensed" font-size="8"
stroke="none" fill="#fff" font-weight="900" style="text-transform:uppercase;letter-
spacing:1px">Gate Sorter A <tspan x="16" dy="17" font-size="12">Step #5</tspan>
                  </text>
                </g>
                <line x1="362" x2="360" y1="50" y2="50" stroke-width="2"
stroke="#443c3d" stroke-dasharray="2,1" />
                <g transform="translate(400)">
                  <rect id="box4" />
                  <text x="14" y="47" font-family="Open Sans Condensed" font-size="12"
stroke="none" fill="#F5F3E7" font-weight="900" style="text-transform:uppercase;letter-
spacing:1px">Sorter <tspan x="12" dy="17" font-size="12">Step #4</tspan>
                  </text>
                </g>
                <line x1="362" x2="400" y1="50" y2="50" stroke-width="2"
stroke="#443c3d" stroke-dasharray="2,1" />
                <g transform="translate(392)">
                  <circle fill="#ffffff" cx="-340" cy="50" r="45" opacity="1" />
                  <text x="-370" y="60" font-family="Open Sans Condensed" font-
size="26" stroke="none" fill="#000" font-weight="100" style="text-
transform:uppercase;letter-spacing:1px">End</text>
                </g>
              </g>
            </svg>
          </div>
        </div>
        <div>
          <div class="flex-container">
            <div class="flex-up">
              <div class="wrp">
                <table border="3">
                  <tr>
```

```
                    <th></th>
                    <th></th>
                    <th></th>
                    <th></th>
                  </tr>
                  <tr>
                    <th>Number Of Iterations</th>
                    <td>
                      <span id="result_12" style="color:#ff0"></span>
                    </td>
                  </tr>
                  <tr>
                    <th>Pass</th>
                    <td>5</td>
                  </tr>
                  <tr>
                    <th>Fail</th>
                    <td>8</td>
                  </tr>
                  <tr>
                    <th>Success Rate</th>
                    <td>67 %</td>
                  </tr>
                  <tr>
                    <th>Iteration Time</th>
                    <td>5 s</td>
                  </tr>
                </table>
              </div>
            </div>
          </div>
        </article>
      </div>
</body>
```