Wydział Elektryczny
POLITECHNIKA WARSZAWSKA

Projekt Zespołowy
Politechnika Warszawska
Wydział Elektryczny
Automatyka i Robotyka Stosowana, II stopień

Przemysłowy system diagnostyczny oparty o wieżę LED RGB

zrealizowany we współpracy z firmą Bosch Rexroth

Raport końcowy

Opiekun: dr inż. Michał Macias
Opiekun ze strony Bosch Rexroth: mgr inż. Ievhen Hrabowskyy

Członkowie zespołu:
inż. Izabela Dusza
inż. Nadia Kerrouche
inż. Magda Koprowska
inż. Kamil Tarachanowicz
inż. Michał Wierzchowski
inż. Mikołaj Wojciuk

Spis treści

1. Cel projektu

Celem projektu było stworzenie przemysłowego systemu diagnostycznego opartego o adresowalne diody LED RGB. Projekt ten powstał w ramach przedmiotu *Projekt Zespołowy*, więc jego celem pośrednim było zapoznanie się z metodykami pracy zespołowej, poznanie od strony praktycznej zagadnień i problemów związanych nie tylko bezpośrednio z problemem badawczym, lecz także pracą zespołową. Dlatego został położony szczególny nacisk na zarządzanie pracą zespołową i w ramach wstępnych działań zostały zdefiniowane cele projektowe, utworzony został podział obowiązków, harmonogram działania i diagramy czasowe. Efektem tego jest niniejsze sprawozdanie, obejmujące treści dotyczące badanego zagadnienia oraz zagadnień związanych z pracą zespołową.

2. Harmonogram prac

2.1 Analiza MoSCoW

Aby lepiej wykonać harmonogram i przygotować się do projektu, w pierwszej kolejności należało przeanalizować cele projektowe. Przeprowadzona została więc analiza wymagań MoSCoW. Dzięki tej metodzie można ustawić priorytet zadań, określając które z nich są najważniejsze i bezwzględnie powinny zostać ukończone (*Must have*), które powinny być zrealizowane ale w razie konieczności można z nich zrezygnować (*Should*), takie których wykonania można się podjąć w przypadku wystąpienia zapasu czasu lecz nie są niezbędne do ukończenia projektu z sukcesem (Could), oraz takie które rozszerzają funkcjonalność projektu, lecz z pewnością nie zostaną podjęte w ramach aktualnego przedsięwzięcia (*Won't*) .Wykonana przez Zespół analiza MoSCoW prezentuje się następująco:

Must have:

1. Dobór odpowiedniego sprzętu (hardware'u)
2. Komunikacja po MQTT - odbieranie błędów od urządzeń zamontowanych na stanowisku Bosch Rexroth
3. Obsługa błędów - alarmy od elementów z szafek
4. Wysterowanie adresowalnych diod LED w zależności od błędów
5. Zapewnienie odpowiedniego zasilania, z zapasem do rozbudowy systemu i niezależnego od innych urządzeń
6. Przetestowanie gotowego systemu
7. Zaakceptowanie systemu przez pracowników firmy Bosch Rexroth

Should have:

1. Możliwie przemysłowe zastosowanie
2. Zaprogramowanie obsługi błędów małoznaczących
3. Komunikacja poprzez IO-Link
4. Użycie komputera przemysłowego PR21
5. Diody zasilane napięciem 24V (mniejszy pobór prądu)

Could have:

1. Zewnętrzny program służący do konfiguracji diod (dodanie kolejnej szafy, skrócenie czy przedłużenie paska, dodanie innych motywów barw świecenia)
2. Uatrakcyjnienie świecenia poprzez wyświetlanie animacji świetlnych zamiast pojedynczych kolorów

WONT:

1. Sterowanie zdalne/bezprzewodowe diodami
2. Dobór jasności świecenia diod do warunków otoczenia
3. Optymalizacja zużycia energii (pobór prądu przy różnej jasności)
4. Załadowywanie paska led podczas startu maszyny niczym kierunkowskazy dynamiczne

W przedstawionej analizie MoSCoW kolorem zielonym oznaczono zadania które zostały zrealizowane bez wprowadzania żadnych zmian, na żółto - zadania które są przeznaczone do dalszego rozwoju, zaś na czerwono - zadania z których zrezygnowano w trakcie trwania projektu.

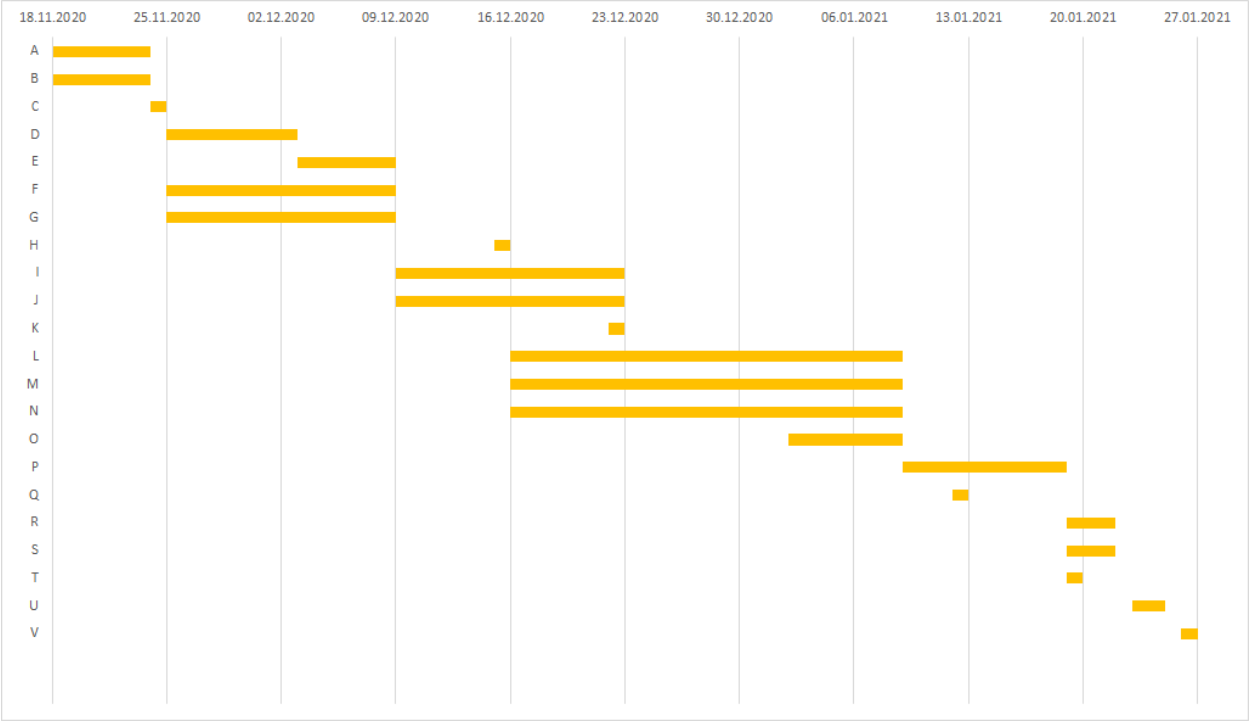
2.2 Wykres Gantta

Czas trwania projektu zaplanowany został w ten sposób, aby zadania zostały równomiernie rozłożone na okres trwania całego semestru. Przy planowaniu każdego zadania zakładano drobny bufor czasowy, tak aby móc zamortyzować ewentualne poślizgi w harmonogramie. Zakładany harmonogram przedstawiony jest w tabeli 0:

Zadanie	Czas trwania [dni]	Zadania poprzedzające	Data rozpoczęcia	Data zakończenia
A	6	-	18.11.2020	24.11.2020
B	6	-	18.11.2020	24.11.2020
C	1	B	24.11.2020	25.11.2020
D	8	C	25.11.2020	03.12.2020
E	6	D	03.12.2020	09.12.2020
F	14	C	25.11.2020	09.12.2020
G	14	C	25.11.2020	09.12.2020
H	1	C	15.12.2020	16.12.2020
I	14	E	09.12.2020	23.12.2020
J	14	E	09.12.2020	23.12.2020
K	1	H	22.12.2020	23.12.2020
L	24	H	16.12.2020	09.01.2021
M	24	H	16.12.2020	09.01.2021
N	24	H	16.12.2020	09.01.2021
O	7	K	02.01.2021	09.01.2021
P	10	O	09.01.2021	19.01.2021
Q	1	K	12.01.2021	13.01.2021
R	3	P	19.01.2021	22.01.2021
S	3	P	19.01.2021	22.01.2021
T	1	Q	19.01.2021	20.01.2021
U	2	R,S,T	23.01.2021	25.01.2021
V	1	U	26.01.2021	27.01.2021

Tabela 0. Harmonogram zadań

Wraz z harmonogramem powstał odpowiadający mu diagram czasowy (tzw. wykres Gantta), ilustrujący dane czynności w czasie. Diagram Gantta dobrze ilustruje sytuacje, w których zadania mogą być wykonywane równolegle i niezależnie od siebie. Wykonany na potrzeby projektu diagram przedstawiono na rysunku 0. Oznaczenia literowe, znajdujące się po jego lewej stronie odnoszą się do poszczególnych procesów (zadań) i zostały opisane w tabeli 1 zamieszczonej pod wykresem.



Rysunek 0. Diagram Gantta

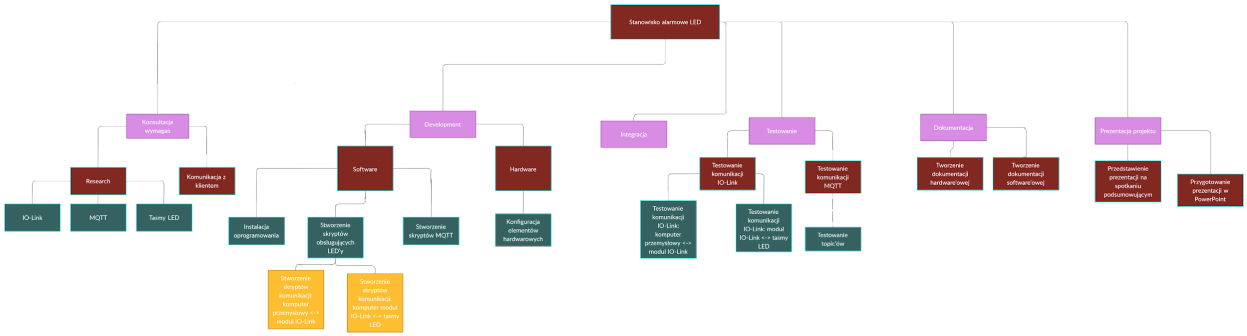
Zadanie	Tytuł	Opis	Osoby odpowiedzialne
A	Konsultacja wymagań		cały zespół
A1	Research IO-Link		Magda, Kamil
A2	Research MQTT		Mikołaj, Iza
A3	Research Taśmy LED		Michał, Nadia
B	Dobór sprzętu na którym realizowany ma być projekt		cały zespół
C	Spotkanie nr 3 z przedstawicielami Bosch Rexroth	zaakceptowanie wybranego zestawu sprzętowego i dyskusja nad tematem	cały zespół
D	Przygotowanie sprzętu	przygotowanie modułu IO-link i wieży IO-link (tymczasowo zamiast paska LED) przez pracowników firmy Bosch Rexrtoth	pracownicy BR
E	Odbiór sprzętu		Michał/Iza

F	Instalacja niezbędnego oprogramowania		cały zespół
G	Założenie repozytorium projektu		Iza
H	Spotkanie nr 4 z przedstawicielami Bosch Rexroth	objaśnianie przykładowych skryptów które uprzednio jeden z przedstawicieli Bosch Rexroth wyśle zespołowi	cały zespół
I	Zainstalowanie stanowiska	na prywatnym komputerze członka zespołu oraz w miarę możliwości, udostępnienie go online	Michał/Iza
J	Dobór konkretnego modelu taśmy LED	która na kolejnym etapie zastąpi wieży IO-link	Magda, Mikołaj
K	Spotkanie nr 5 z przedstawicielami Bosch Rexroth	dyskusja nad postępem prac, zamówienie docelowej taśmy LED	cały zespół
L	Testowanie sprzętu	pisanie skryptow obsługujących sterowanie wieżą IO-link	Michał, Iza, Mikołaj
M	Testowanie sprzętu	opracowywanie modelu komunikacji MQTT	Magda,Nadia,Kamil
N	Opracowanie komunikacji ze sprzętem znajdującym się w firmie	utworzenie odpowiednich "topic'ów" protokołu MQTT wystawianych przez urządzenia znajdujące się stacjonarnie na stanowisku	Ievhen (BR)
O	Odbiór docelowej taśmy LED	odbiór taśmy i integracja z modułem IO-link	Michał/Magda/Iza
P	Napisanie docelowego oprogramowania	pisanie skryptów nasłuchujących urządzenia po MQTT oraz sterujących LEDami	cały zespół
P1	Stworzenie skryptów obsługujących LED'y (1)	Stworzenie skryptów komunikacji: komputer przemysłowy <-> moduł IO-Link	Magda, Kamil
P2	Stworzenie skryptów obsługujących LED'y (2)	Stworzenie skryptów komunikacji: komputer moduł IO-Link <-> taśmy LED	Michał, Nadia
P3	Stworzenie skryptów MQTT		Mikołaj, Iza
Q	Spotkanie nr 6 z przedstawicielami Bosch Rexroth	dyskusja nad postępem prac	cały zespół
R	Przeprowadzanie testów na gotowym systemie	i systematyczne wprowadzanie poprawek	Michał, Iza, Mikołaj
S	Przygotowywanie dokumentacji końcowej		Magda, Nadia, Kamil
T	Spotkanie nr 7 z przedstawicielami Bosch Rexroth	ostateczne zaakceptowanie projektu przez Bosch Rexroth	cały zespół
U	Przygotowywanie prezentacji końcowej		Iza, Magda, Nadia, Kamil
V	Prezentacja końcowa i ostateczne oddanie projektu		Mikołaj, Michał

Tabela 1. Szczegółowy opis harmonogramowanych zadań

2.3 Struktura Podziału Pracy

Oprócz harmonogramu oraz diagramu Gantta stworzyliśmy Strukturę Podziału Pracy (ang. *WBB - Work Breakdown Structure*), stosując metodę zstępującą. Wyszliśmy z najwyższego poziomu projektu, a następnie podzieliliśmy je na fazy, które następnie zostały rozdzielone na etapy. Ostatni poziom stanowią zadania, które zostały wyszczególnione z tych etapów.



3. Wykorzystane narzędzia

Projekt zespołowy, szczególnie realizowany zdalnie wymaga stosowania pewnych narzędzi, który służyły do komunikacji zarówno zespołowej oraz z opiekunem projektu. Projekt ten był projektem związanym z programowaniem dlatego potrzebowaliśmy odpowiedniej aplikacji do przechowywania i współpracy na repozytorium. Lista wszystkich użytych narzędzi i przeznaczenie opisane są poniżej.

3.1. Narzędzia pracy zespołowej

Jako główne narzędzia do zdalnego prowadzenia projektu wykorzystywane były:

- **Microsoft Teams** - narzędzie do kontaktów z kadrą nauczycielską,
- **Facebook Messenger** - narzędzie do szybkiej komunikacji w sprawach bieżących pomiędzy zespołem,
- **Google Docs** - narzędzie do tworzenia raportu i przechowywania notatek
- **Skype** - narzędzie do przeprowadzania spotkań z przedstawicielami firmy Bosch Rexroth

3.2. Narzędzia programistyczne

Projekt wymagał utworzenia i zarządzania kodem programów, w tym celu wykorzystywane były:

- **GitHub** - przestrzeń do przechowywania i współdzielenia kodu
- **Google Drive** - przestrzeń do przechowywania wszystkich dokumentów i plików związanych z projektem
- Wybrane edytory tekstu i kodu takie jak **Sublime Text**, **PyCharm**, **Spider**, **Notepad++**
- **IndraWorks** - oprogramowanie wykorzystywane do tworzenia kodu na sterownik PLC dostarczone przez Bosch Rexroth
- **JanzTec** - biblioteka do IndraWorks umożliwiająca komunikację z wykorzystaniem MQTT z poziomu sterownika PLC
- **Mosquitto** - oprogramowanie opensource pełniące rolę brokera w komunikacji MQTT

W tym miejscu należy wspomnieć także o wybranym języku programowania. Zdecydowano się na wybór języka skryptowego Python, ze względu na przyjazną formę kodu, zrozumiałą dla osób na różnym stopniu zaawansowania oraz dostęp do wielu bibliotek, rozszerzeń, edytorów. Dodatkowym czynnikiem przemawiającym za Pythonem jest bogactwo artykułów i wpisów na temat narzędzi i funkcjonalności tego języka na blogach i forach internetowych.

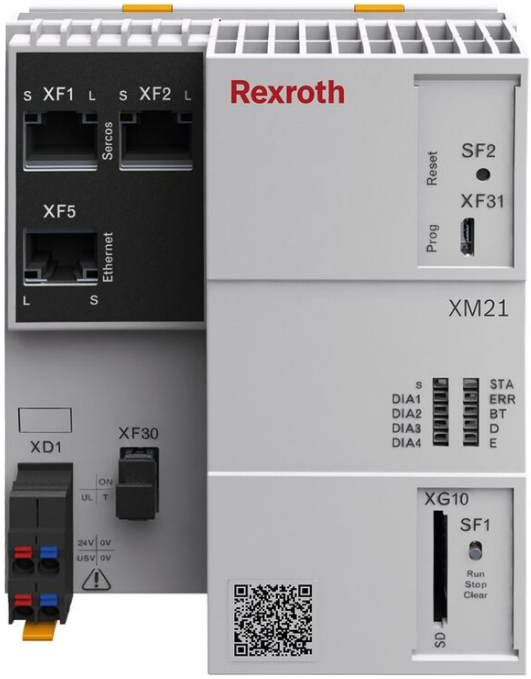
4. Wykorzystany sprzęt

Sprzęt wykorzystywany w projekcie został dostarczony przez przedstawicieli firmy Bosch Rexroth. Zestaw składał się z następujących części:

a. Stanowisko PLC
   - Wyłącznik nadprądowy
   - Zasilacz
   - Sterownik XM21
   - Moduł RT-Ethernet

b. IO-Link
   - Wyspa IO-Link BNI004U
   - Lampa BNI0084
   - Przewód zasilający wyspę
   - Przewód wyspa-XM21
   - Przewód wyspa-lampa

c. Przewód Zasilający
d. Przewód Ethernet: 2 sztuki



Rysunek 1. Sterownik XM21 wykorzystany w projekcie

Rysunek 2. Wyspa IO-Link BNI004U oraz lampa BNI004 wykorzystane w projekcie.

5. Wymagania postawione przez firmę Bosch Rexroth

Przedstawiciele firmy Bosch Rexroth na początku realizacji projektu postawili przed członkami zespołu zbiór wymagań, które w toku pracy nad projektem okazały się niemożliwe do zrealizowania ze względu na ograniczenia w dostępie do linii pokazowej Bosch Rexroth, jak również ograniczenia czasowe projektu.

Pierwotne wymagania prezentowały się następująco:

1. Porównanie dostępnych produktów i rozwiązań na rynku
2. Projekt układu taśm LED adresowalnych
3. Wyspecyfikowanie produktów
4. Skrypt do sterowania układem taśm LED
    a. Język programowania Python
5. Komunikacja z programem przy użyciu protokołu Mqtt
6. Sterowanie diodami LED z wejść/wyjść komputera przemysłowego lub dedykowanego sterownika.

7. Skalowalność, możliwość dodania w prosty sposób kolejnej sekcji LED.
8. Obsługa błędów (np. zerwanie połączenia z brokerem Mqtt)
9. Skrypt do obsługi poleceń (np. czytający błędy maszyn)
    a. Język programowania Python
10. Komunikacja z programem przy użyciu protokołu Mqtt
11. Skalowalność, możliwość dodania w prosty sposób kolejnego urządzenia zgłaszającego awarię.
12. Obsługa błędów (np. zerwanie połączenia z brokerem Mqtt)
13. Przeprowadzenie testów
14. Sporządzenie dokumentacji oraz instrukcji obsługi
15. Implementacja na linii dydaktyczno-pokazowej Showroom firmy Bosch Rexroth

Po kilku spotkaniach z przedstawicielami firmy doszliśmy do wymagań, które były możliwe do realizowania. Zostały one sformułowane w poniższy sposób:

1. Wymyślenie statusów linii produkcyjnej, które obsługiwać będzie lampa,
2. Wymyślenie struktury tematów MQTT,
3. Zainstalowanie brokera MQTT,
4. Stworzenie programu na sterownik PLC sterującego lampą,
5. Napisanie skryptu nadającego wiadomości z wykorzystaniem MQTT,
6. Konfiguracja sterownika PLC do obsługi MQTT,
7. Integracja skryptów napisanych w języku Python z programem PLC sterującym wieżą LED poprzez protokół MQTT.

Ponadto, stworzone programy miały być proste w wykorzystaniu, implementowane w warunkach przemysłowych oraz skalowalne.

6. Wprowadzenie do zagadnienia

Projekt polegał na stworzeniu przemysłowego systemu diagnostycznego opartego o wieżę LED RGB. System diagnostyczny bazował na komunikacji pomiędzy sterownikiem PLC a wieżą LED poprzez wyspę IO-Link przy wykorzystaniu brokera Mosquitto oraz protokołu komunikacyjnego MQTT.

MQTT jest protokołem komunikacyjnym opierającym się na modelu publikowania i subskrypcji. Wyróżnić można w nim kilka ról, z których to kluczową odgrywa tak zwany broker. Jest to oprogramowanie zainstalowane na danym komputerze, które odpowiada za pracę systemu komunikacji. Do jego zadań należy przyjmowanie i przekazywanie wiadomości zgodnie z ustaloną hierarchią tematów, przyjmowanie połączeń od nowych urządzeń, autoryzacja podłączonych klientów czy w końcu ogólna konfiguracja sposobu komunikacji.

W systemie oprócz brokera znajdują się także klienci. Klientem może być urządzenie lub też na przykład pracujący program, nawet na tej samej maszynie co broker. Klienci (w zależności od konfiguracji brokera) muszą posiadać swoją nazwę i hasło. Od brokera zależy, czy dany klient będzie mógł subskrybować, publikować czy wykonywać obie czynności na danym “topicu” (temacie). Wspomniany temat jest to swego rodzaju kanał informacyjny, na którym mogą publikować wiadomości klienci a inni klienci mogą je odczytywać.

Dzięki podziałowi na tematy, podłączyć do sieci można wiele urządzeń a następnie określić, które z nich mają mieć dostęp do danych informacji i kanałów komunikacji. Komunikacja MQTT jest bardzo elastyczna, nie ma ograniczeń co do formy klientów (czy jest to urządzenie mobilne, mikrokontroler, sterownik PLC) oraz jest łatwo skalowalna. Działa w sieci lokalnej, jej konfiguracja jest relatywnie prosta i oferuje duże możliwości konfiguracji. Ideą stojącą poniekąd za MQTT jest Internet Rzeczy, koncept w którym to wiele urządzeń podłączonych jest do jednej sieci i zachodzi ciągła wymiana danych pomiędzy urządzeniami.

Przykładem zastosowania MQTT może być system inteligentnego domu, gdzie czujniki temperatury w pomieszczeniach wysyłają informacje do brokera, ten przekazuje te dane do jednostki odpowiadającej za logikę, która z kolei wysyła informacje dalej, do urządzeń wykonawczych typu grzejniki. W omawianym projekcie skomunikowanymi urządzeniami są komputer z brokerem MQTT (broker Mosquitto) oraz sterownik PLC.

Sterownik PLC, z wykorzystaniem biblioteki Janz_tec_MQTT, łączy się z brokerem MQTT i subskrybuje odpowiednie tematy. Program napisany w języku Python służy za odrębnego klienta, który publikuje wiadomości na kanał tematu, który odbiera sterownik PLC. Komunikacja przechodzi przez brokera, a sterownik po odczytaniu wiadomości, wykonuje określone czynności.

Do sterownika PLC, poprzez wyspę I/O Link, podłączona jest wieża sygnalizacyjna posiadająca kilka segmentów z diodami RGB. Sterownik PLC, po odczytaniu treści otrzymanych wiadomości w ustalony sposób steruje lampą sygnalizacyjną. W ten sposób możliwe jest sterowanie kolorami i segmentami diod na wieży z poziomu programu napisanego w Pythonie. Ideą stojąca za tym rozwiązaniem jest symulacja wyświetlania informacji (alertów, alarmów) pochodzących z poszczególnych komponentów składowych

systemu automatyki przemysłowej, zdolnych do komunikacji w formacie MQTT. Rozwiązanie takie ma duży potencjał na praktyczne wykorzystanie, pozwala on bowiem agregować informacje pochodzące z różnych miejsc na linii produkcyjnej i w zależności od tych informacji podejmować różne decyzje. Dodatkowo, komunikacja poprzez MQTT oferuje duże możliwości akwizycji danych celem ich późniejszej analizy, co może być przydatne w przypadku potrzeby znalezienia awarii czy usprawnienia działania linii produkcyjnej.

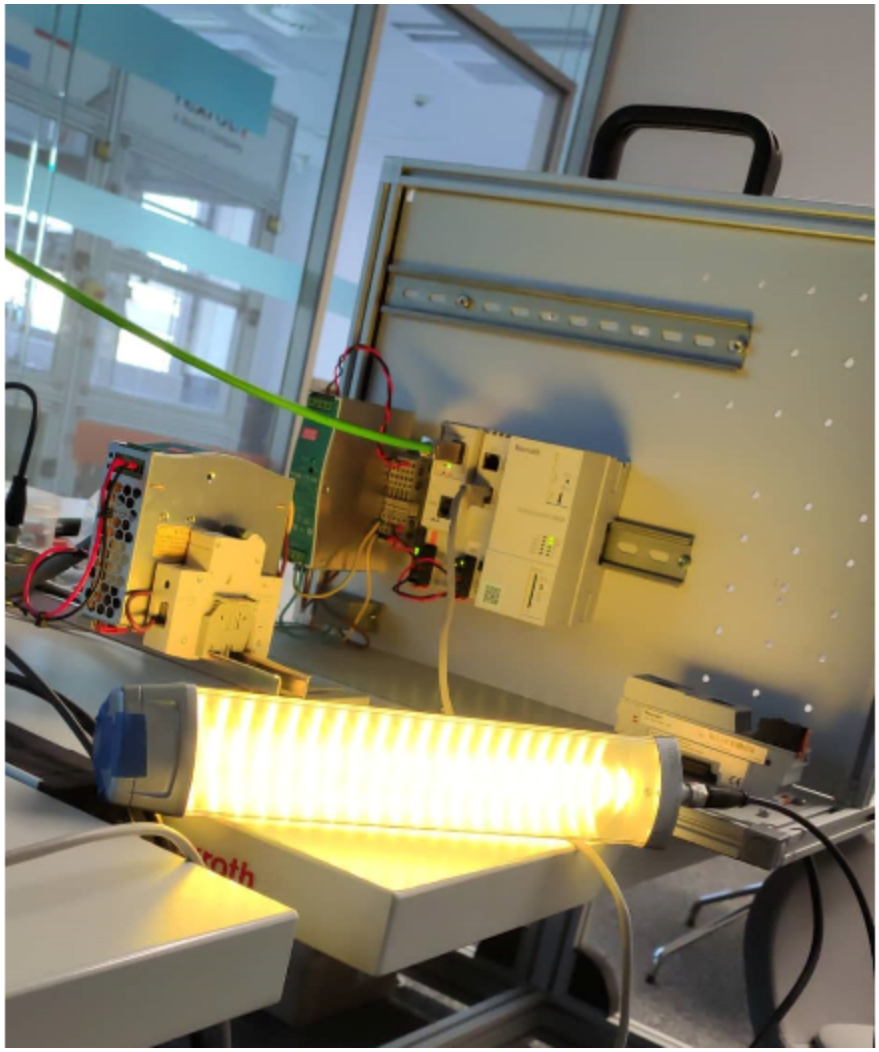
7. Realizacja projektu

Projekt składał się z dwóch części - części sprzętowej (hardware) oraz części programistycznej (software).

7.1 Hardware

Część sprzętowa polegała na uruchomieniu stanowiska składającego się ze sterownika XM21, wieży IO-Link BNI004U, lampy BNI004U firmy Balluff oraz elementów potrzebnych do ich zasilenia oraz skomunikowania. Stanowisko zostało odebrane z firmy Bosch Rexroth przez członka zespołu. Poniżej zostały przedstawione zdjęcia przekazanego stanowiska.

Rysunek 3. Stanowisko odebrane od firmy Bosch Rexroth - sterownik XM21, wyspa IO-Link BNI004U oraz lampa BNI004.



Rysunek 4. Test lampy BNI004 w firmie Bosch Rexroth przed przekazaniem sprzętu zespołowi.

Wieża składa się z 5 segmentów, jednak dla celów projektu wykorzystywane będą 3. Każdy z segmentów może świecić się na 7 różnych kolorów. Każdy segment sterowany jest 4 bitami, odpowiednio dla kolorów zielonego, czerwonego i niebieskiego oraz kontroli migania. Za sterowanie wieżą odpowiada sterownik PLC podłączony do wyspy I/O Link oraz komputera PC. Port XF5 sterownika XM21 odpowiedzialny jest za połączenie z komputerem i ma skonfigurowany adres IP na wartość 192.168.1.1. Dlatego aby połączenie było możliwe należy skonfigurować adres IP portu Ethernet komputera na dowolną niezajętą wartość z rodziny 192.168.1.xxx. W jednostce używanej w projekcie wartość ta ustawiona była na 192.168.1.100.

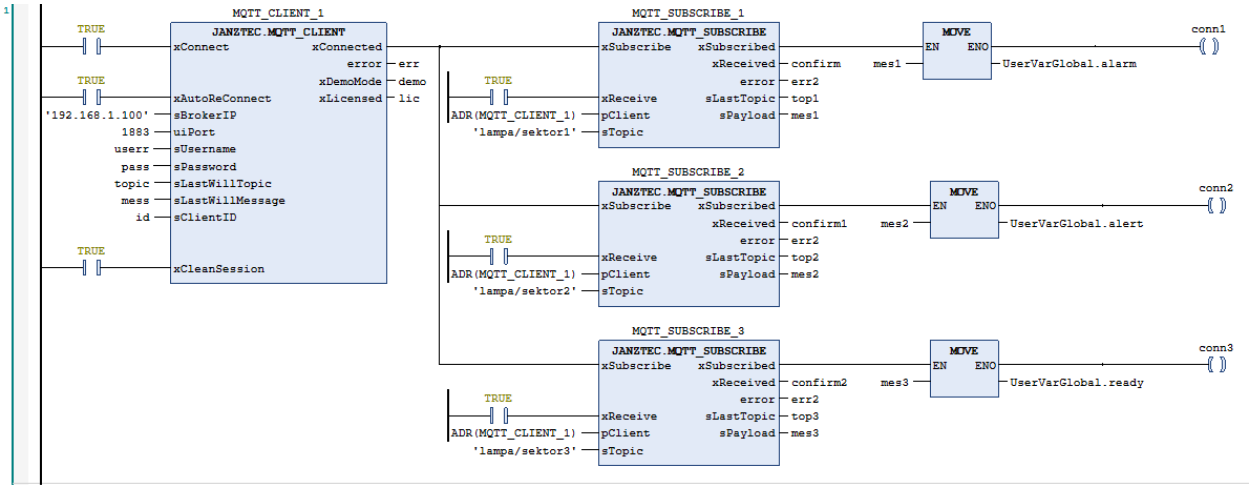
7.2 Software

Część programistyczna projektu składała się z napisania oprogramowania na sterownik PLC w języku drabinkowym oraz ST. Powstały także programy w języku Python służące do komunikacji ze sterownikiem PLC i pośrednio z wieżą LED.

7.2.1 Programy PLC

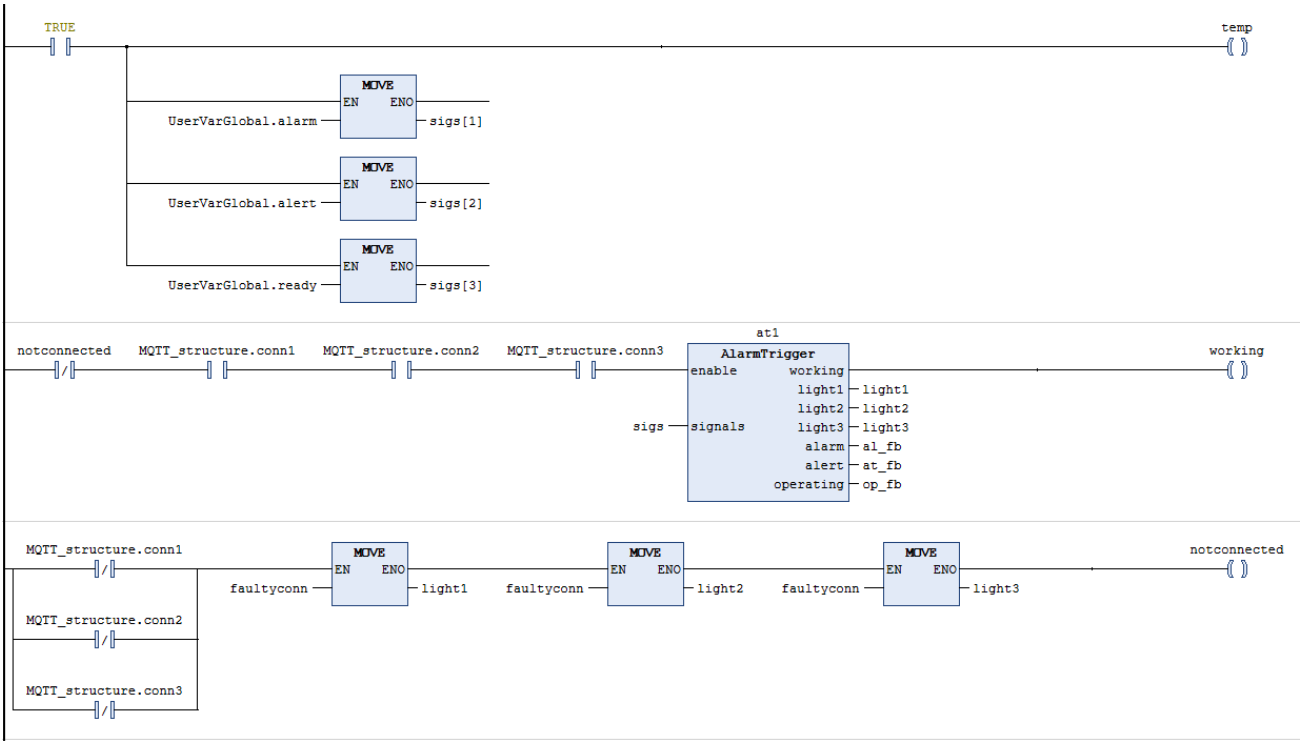
Programy na sterownik XM21 został napisany w środowisku IndraWorks Engineering. Wykorzystując bibliotekę JanzTec, sterownik ten mógł komunikować się z wykorzystaniem MQTT. Idea programu opiera się na odbieraniu danych pochodzących z kanału MQTT i przetworzenie ich, by następnie w adekwatny sposób kontrolować wieżę LED (jej kolory świecenia i miganie).

Pierwszy program służy do komunikowania zdarzeń na maszynie. Obsługiwane są trzy ich typy: alarmy (zdarzenia krytyczne uniemożliwiające poprawną pracę maszyny), alerty (zdarzenia niepożądane, przy których maszyna może nadal pracować) oraz stan maszyny (czy maszyna pracuje czy też czeka na uruchomienie).



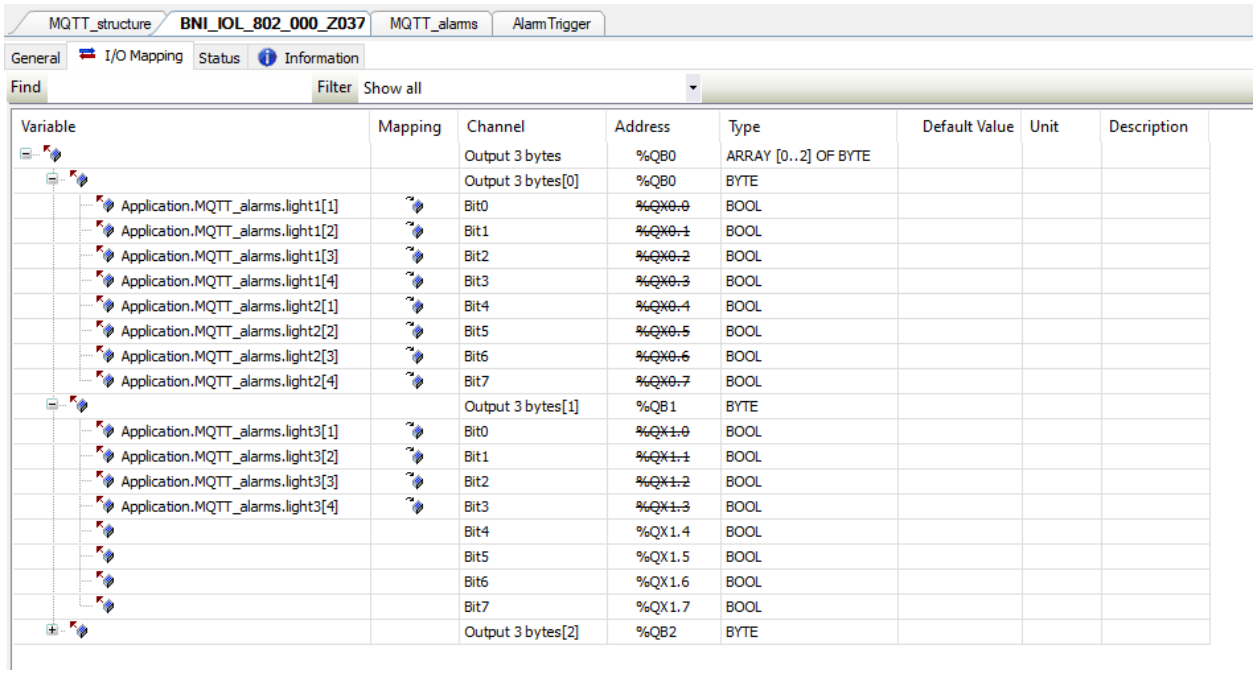
Rysunek 5. Bloki funkcyjne odpowiadające za utworzenie klienta i subskrypcję

Każdy z trzech segmentów lampy subskrybuje inny temat. Umożliwia nam to niezależną kontrolę nad nimi. Jeśli połączenie z klientem się powiodło oraz dany temat został zasubskrybowany, to otrzymane wiadomości przepisywane są do globalnych zmiennych odpowiedzialnych za alarm, alert oraz stan działania maszyny i wystawiane są sygnały gotowości każdego z segmentów.



Rysunek 6. Obsługa wiadomości otrzymanych przez MQTT

Pierwsza linia kodu odpowiada tylko za przepisywanie otrzymanych wiadomości do jednej tablicowej zmiennej globalnej. Druga linia odpowiada za interpretację otrzymanych wiadomości i obsługuje wieżę. Za ten proces odpowiedzialny jest blok funkcyjny “AlarmTrigger”, który dokonuje odczytu sygnałów ze zmiennej lokalnej “sigs”, a następnie zwraca macierzowe zmienne wartości logicznych “light1”, “light2” oraz “light3”, odpowiedzialne za sterowanie wieżą. W trzeciej linii sprawdzany jest status połączenia z tematami. Jeśli którykolwiek z nich nie jest połączony, to cała wieża mruga na biało.

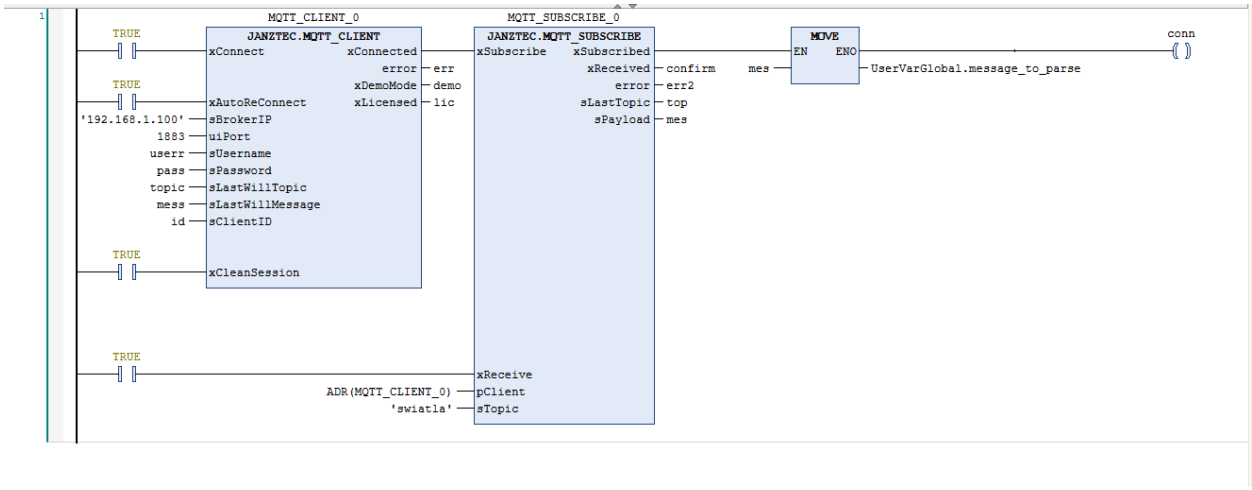


Variable	Mapping	Channel	Address	Type	Default Value	Unit	Description
		Output 3 bytes	%QB0	ARRAY [0..2] OF BYTE			
		Output 3 bytes[0]	%QB0	BYTE			
Application.MQTT_alarms.light1[1]		Bit0	~~%QX0.0~~	BOOL			
Application.MQTT_alarms.light1[2]		Bit1	~~%QX0.1~~	BOOL			
Application.MQTT_alarms.light1[3]		Bit2	~~%QX0.2~~	BOOL			
Application.MQTT_alarms.light1[4]		Bit3	~~%QX0.3~~	BOOL			
Application.MQTT_alarms.light2[1]		Bit4	~~%QX0.4~~	BOOL			
Application.MQTT_alarms.light2[2]		Bit5	~~%QX0.5~~	BOOL			
Application.MQTT_alarms.light2[3]		Bit6	~~%QX0.6~~	BOOL			
Application.MQTT_alarms.light2[4]		Bit7	~~%QX0.7~~	BOOL			
		Output 3 bytes[1]	%QB1	BYTE			
Application.MQTT_alarms.light3[1]		Bit0	~~%QX1.0~~	BOOL			
Application.MQTT_alarms.light3[2]		Bit1	~~%QX1.1~~	BOOL			
Application.MQTT_alarms.light3[3]		Bit2	~~%QX1.2~~	BOOL			
Application.MQTT_alarms.light3[4]		Bit3	~~%QX1.3~~	BOOL			
		Bit4	%QX1.4	BOOL			
		Bit5	%QX1.5	BOOL			
		Bit6	%QX1.6	BOOL			
		Bit7	%QX1.7	BOOL			
		Output 3 bytes[2]	%QB2	BYTE			

Rysunek 7. Sterowanie wieżą LED

Na rysunku 7. przedstawiona jest konfiguracja odpowiednich bitów wieży. Elementy zmiennych tablicowych przechowujących informację o kolorach i mruganiu przypisane są odpowiednim wartościom w bloku GSDML.

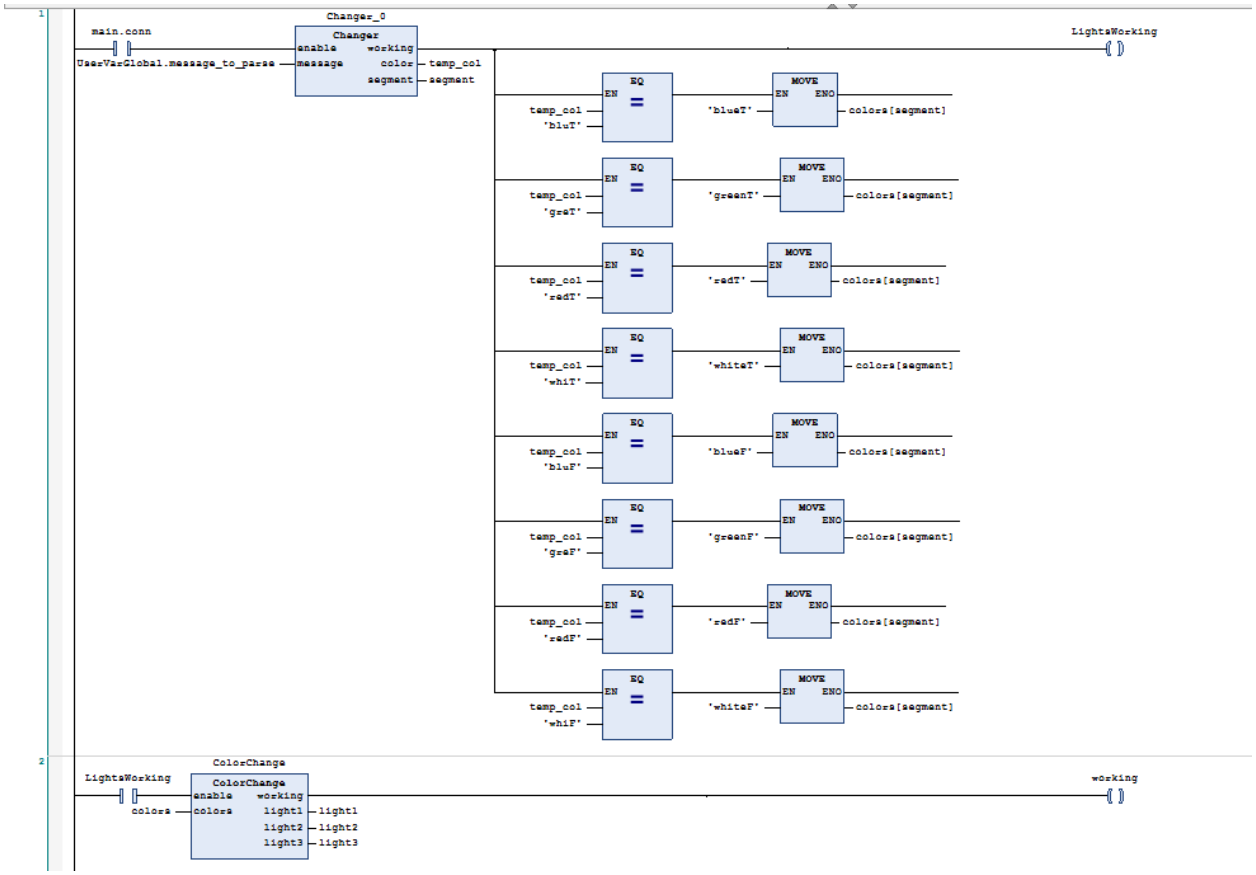
Drugi program odpowiedzialny jest za możliwość zmiany koloru wybranego segmentu manualnie przez użytkownika. Schemat połączenia z MQTT przedstawiony jest na rysunku 8.



Rysunek 8. Bloki funkcyjne odpowiadające za utworzenie klienta i subskrypcję

W przeciwieństwie do programu pierwszego, w tym przypadku występuje tylko jeden temat, na którym publikowane są wiadomości. Otrzymana wiadomość przekazywana jest do zmiennej globalnej “message_to_parse”.

Interpretacja otrzymanych wiadomości przedstawiona jest na rysunku 9.



Rysunek 9. Obsługa wiadomości otrzymanych z MQTT

Jeśli połączenie z klientem i subskrypcja powiodły się, to otrzymywane wiadomości interpretowane są w bloku funkcyjnym “Changer”. Blok ten zwraca zmienną “color” typu String oraz zmienną “segment” typu Integer. Następnie wartości te porównywane są z oczekiwanymi i wpisywane do odpowiedniego elementu w tablicy zmiennych logicznych “colors”. Następnie w bloku funkcyjnym “ColorChange” wartości te są odpowiednio przypisywane do odpowiadających im w konfiguracji bloku GSDML miejsc, analogicznie jak w tej przedstawionej na rysunku 7.

7.2.2 Broker MQTT

Kod brokera MQTT sprowadza się do utworzenia pliku konfiguracyjnego dla brokera. Obejmuje on liczne funkcje i parametry, które można dowolnie konfigurować. Dla potrzeb projektu wykonana została krótka konfiguracja, umożliwiająca połączenie się z brokerem i komunikację pomiędzy urządzeniami. W celu dodania autoryzacji klientów czy wprowadzeniu dodatkowych ustawień, należy zmodyfikować plik z rozszerzeniem .conf dostępny w katalogu instalacyjnym brokera Mosquitto.

7.2.3 Programy w języku Python

Rolą pierwszego kodu jest generowanie i nadawanie odpowiednio przygotowanych wiadomości. Wiadomości te generowane są na podstawie symulowanych zdarzeń i są formatowane tak, by były możliwe do odczytania i wykorzystania przez sterownik PLC. Na kod składa się przede wszystkim utworzenie instancji klienta (z wykorzystaniem biblioteki PAHO), konfiguracja połączenia z brokerem pod odpowiednim adresem sieci lokalnej oraz zasadnicze przygotowanie i transmisja wiadomości. Struktura wiadomości jest tworzona w taki sposób, by finalnie na lampie podłączonej do sterownika PLC wyświetlanych było kilka różnych schematów.

Stan	**Góra**	**Środek**	**Dół**
1.	Nie świeci	Zielony świeci	Zielony świeci
2.	Nie świeci	Zielony świeci	Zielony miga
3.	Nie świeci	Pomarańczowy miga	Zielony świeci
4.	Nie świeci	Pomarańczowy miga	Zielony miga
5.	Czerwony miga	Czerwony miga	Czerwony miga

Tabela 2. Symulowane stany maszyny

Góra, środek i dół odnoszą się do trzech segmentów na wieży z diodami. Tabela przedstawia cztery schematy świecenia, ustawiane w zależności od wygenerowanych przez program wiadomości. Schematy te odpowiadają następującym, symulowanym, stanom maszyny:

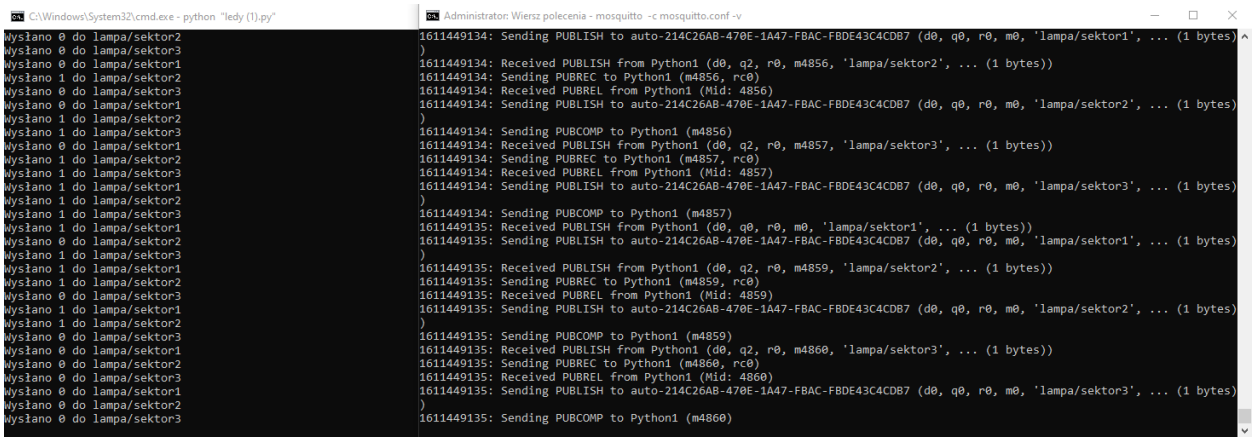
1. Sprawna, oczekuje
2. Sprawna, pracuje
3. Alert, oczekuje
4. Alert, pracuje
5. Niesprawna, alarm

Aby wygenerować wiadomość z poziomu programu, należy wcisnąć odpowiedni przycisk na klawiaturze:

- Przycisk “a” - alarm
- Przycisk “s” - alert
- Przycisk “d” - stan maszyny

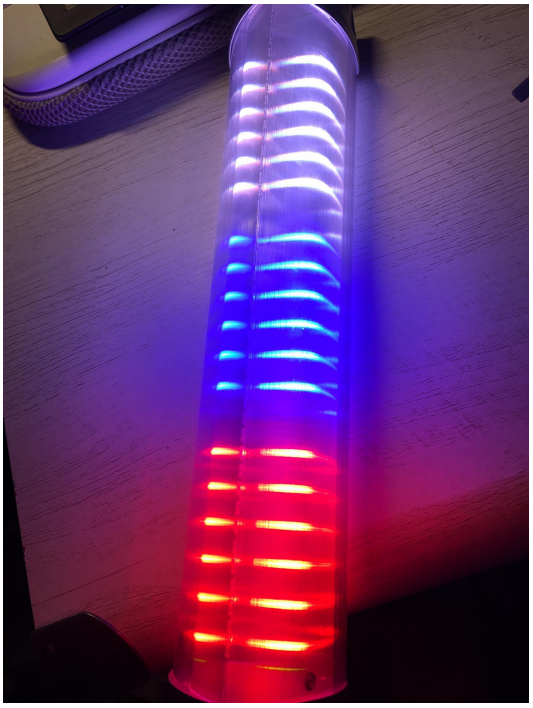
Nadmienić należy, że stan alarmowy jest nadrzędny w stosunku do pozostałych stanów - jeśli jest on wystawiony, to wyświetlanie informacji o alertach / stanie maszyny nie będzie możliwe, dopóki alarm nie zniknie.

Utworzony został także dodatkowy program, służący do ręcznej konfiguracji kolorów i migania poszczególnych segmentów wieży. Kod ten wymaga wpisania odpowiedniej formuły, opisującej kolor, numer segmentu oraz czy dany segment ma migotać. Jeżeli użytkownik wpisze wiadomość, “green 1 T”, to program wyśle wiadomość “gre1T” co odpowiada zmianie koloru na segmencie górnym na migający zielony, a na przykład wiadomość “blue 2 N” spowoduje wysłanie komunikatu “blu2F” ustawi na środkowym segmencie kolor niebieski świecący stałym światłem. Jeśli użytkownik wpisze nieinterpretowaną przez program wartość koloru, segmentu i instrukcji mrugania, to domyślnie wysłana zostanie wiadomość “whi1F”, która spowoduje ustawienie koloru białego świecącego stałym światłem na górnym segmencie wieży. Rysunek 10. przedstawia przykładowy stan komunikacji między skryptem w języku Python generującym zdarzenia na maszynie, a brokerem Mosquitto.



Rysunek 10. Stan komunikacji z brokerem MQTT
(po lewej nadawanie z programu, po prawej okno podglądu stanu brokera MQTT)

Rysunek 11. Świecenie wieży w zależności od otrzymanej wiadomości (stany 1.,3.,5.)



Rysunek 12. Świecące segmenty na skutek wysłanych, ręcznie wprowadzonych wiadomości

```
    color = input('Wprowadź kolor (red, green, blue): ')
KeyboardInterrupt
^C
D:\michw\Nowy folder\IO-Link>

D:\michw\Nowy folder\IO-Link>python swiatla_manual.py
Nawiązywanie połączenia
Wprowadź kolor (red, green, blue): red
Zła nazwa koloru!
Wprowadź numer segmentu (od 1 do 3): 3
Czy segment ma mrugać?: (T/N)N
Wysłano  whi3F do tematu swiatla
Wprowadź kolor (red, green, blue): red
Wprowadź numer segmentu (od 1 do 3): 2
Czy segment ma mrugać?: (T/N)T
Wysłano  red2T do tematu swiatla
Wprowadź kolor (red, green, blue): blue
Wprowadź numer segmentu (od 1 do 3): 1
Czy segment ma mrugać?: (T/N)N
Wysłano  blu1F do tematu swiatla
Wprowadź kolor (red, green, blue): blue
Wprowadź numer segmentu (od 1 do 3): 2
Czy segment ma mrugać?: (T/N)N
Wysłano  blu2F do tematu swiatla
Wprowadź kolor (red, green, blue): wrong
Zła nazwa koloru!
Wprowadź numer segmentu (od 1 do 3): 0
Zły segment!
Czy segment ma mrugać?: (T/N)k
Wysłano  whi1F do tematu swiatla
```

Rysunek 13. Przykładowe ręczne sterowanie segmentami wieży LED z poziomu programu w Pythonie

8. Kosztorys

Klasyfikacja kosztów w projekcie:

- koszty pośrednie: -,
- koszty bezpośrednie: sprzęt potrzebny do zrealizowania celu projektu,
- koszty stałe: wynagrodzenia pracowników,
- koszty zmienne: -,
- ze względu na charakterystykę zasobów projektu: zużycie materiałów i energii.

Wykorzystana metoda do oszacowania kosztów w projekcie:

- Estymowanie od dołu do góry: podział na poszczególne zadania i sprzęt oraz pozostałe zasoby potrzebne do ich realizacji. Ostateczna estymata kosztów to suma z poszczególnych zadań.

Koszty bezpośrednie:

- Wyspa IO-Link BNI004U: 549.00 €
- Lampa BNI0084: 430.00 €

Metoda opracowania budżetu:

- w oparciu o przygotowany harmonogram (zasoby i zadania): Jaki sprzęt jest niezbędny do realizacji konkretnych zadań w projekcie, na jakim sprzęcie będziemy pracować, aby uzyskać zamierzony cel projektu.

9. Podsumowanie

9.1. Możliwości rozwoju projektu

Wykonany projekt posiada duży potencjał rozwojowy. Przede wszystkim, system w docelowej wersji może zostać zaimplementowany na linii produkcyjnej. System zainstalowany na produkcji powinien w jasny i klarowny sposób w czasie rzeczywistym informować operatorów linii o błędach i awariach. Narzędzie alarmujące o awariach są bardzo ważne i przyspieszają czas reakcji operatora, dzięki czemu pomagają zapobiegać występowaniu poważnych konsekwencji.
W wykonanym systemie możliwe jest zastosowanie innych komponentów rozszerzających funkcjonalność systemu lub obniżających koszty jego wykonania. Dla przykładu, zastosowane kolumny LED RGB można by było zastąpić taśmami LED. Takie rozwiązanie byłoby tańsze, ale wymagałoby zastosowania odpowiednich przetworników i zaktualizowania oprogramowania.

9.2. Podsumowanie pracy zespołowej

Mimo teoretycznych założeń dotyczących harmonogramu, po drodze do zrealizowania celu wielokrotnie musieliśmy mierzyć się z różnego rodzaju poślizgami czasowymi oraz trudnościami sprzętowymi. W trakcie współpracy pojawiały się też konflikty, wynikające z różnych wizji poprowadzenia projektu, jednak dzięki licznym rozmowom udało nam się dojść do porozumienia w każdym aspekcie. Projekt zakończył się powodzeniem, założone na początku zadania udało się wykonać. Harmonogram prac był przestrzegany przez cały okres, a postępy były dokumentowane w formie sprawozdań dostarczanych w wyznaczonym terminie. Każdy z członków z zespołu wykonał swoją pracę na czas i w bardzo dobrej jakości. Wykorzystane narzędzia do pracy zespołowej zostały dobrane odpowiednio, ale przy pracy w większym zespole wybralibyśmy raczej bardziej zaawansowane aplikacje. Bez wątpienia projekt ten był bardzo ciekawym doświadczeniem, z którego jako zespół, wynieśliśmy wiele cennych uwag i wniosków dotyczących pracy zespołowej.

Podsumowując, dzięki uczestnictwie w niniejszym projekcie poznaliśmy i doceniliśmy narzędzia do zarządzania wiedzą. Zrozumieliśmy, że do wykonania projektu niezbędne jest zebranie odpowiednich celów i założeń projektowych na początkowym etapie, określenie zadań i przypisanie do każdego z nich osoby odpowiedzialnej za jego wykonanie oraz oszacowanie czasu niezbędnego do jego ukończenia. Ponadto, doceniliśmy siłę harmonogramowania zadań. Bez ścisłego trzymania się zaplanowanych terminów wykonania zadań, zarządzanie projektem kilkuosobowego zespołu byłoby bardzo trudne, a być może wręcz niemożliwe.