Wrocław, 7 lutego 2014

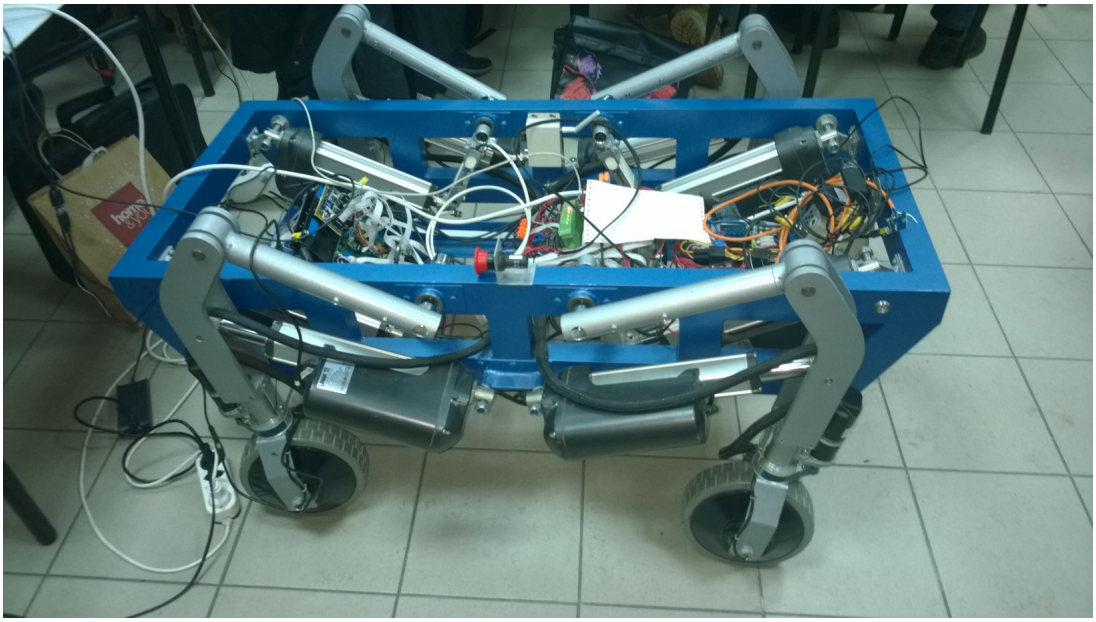
Dokumentacja projektu

System sterowania robota LegVan

Wydział Elektroniki
Kierunek: Automatyka i Robotyka
Specjalność: Robotyka

Spis treści

1 Robot LegVan II



Rysunek 1: Robot LegVan II

Robot LegVan II to platforma mobilna. Może być stosowana do celów transportowych oraz badawczych. Celem projektu było stworzenie opracowanie sterownika robota, modułu zadajnika oraz przygotowanie aplikacji do akwizycji danych.Sterownik robota pozwala na wczytanie skryptów implementujących algorytm sterowania robota, implementację podstawowych komend sterujących. Wykorzystano do tego platformę programistyczną URBI. Założenia modułu zadajnika to możliwość wykorzystania różnych zadajników, sterowanie robotem za ich pomocą oraz przygotowanie struktury danych konwertera. Aplikacja do akwizycji danych odpowiaada za pobranie danych z robota ma ułatwiać użytkowanie robota oraz wizualizację jego stanu na podstawie pobranych danych. Cele projektu udało się zrealizować niemal w pełni. Podczas całości trwania projektu wykonano aplikację diagnostyczną, pozwalającą na śledzenie położenia robota, odczytu z sensorów czy z kamery. Dodatkowo został stworzony moduł przesyłu komend do robota za pomocą zadajnika oraz poprzez urbiscript. Stworzono komendy niskiego oraz średniego poziomu. Poniższa dokumentacja prezentuje po krótce możliwości stworzonej apikacji diagnostycznej oraz sterownika robota.

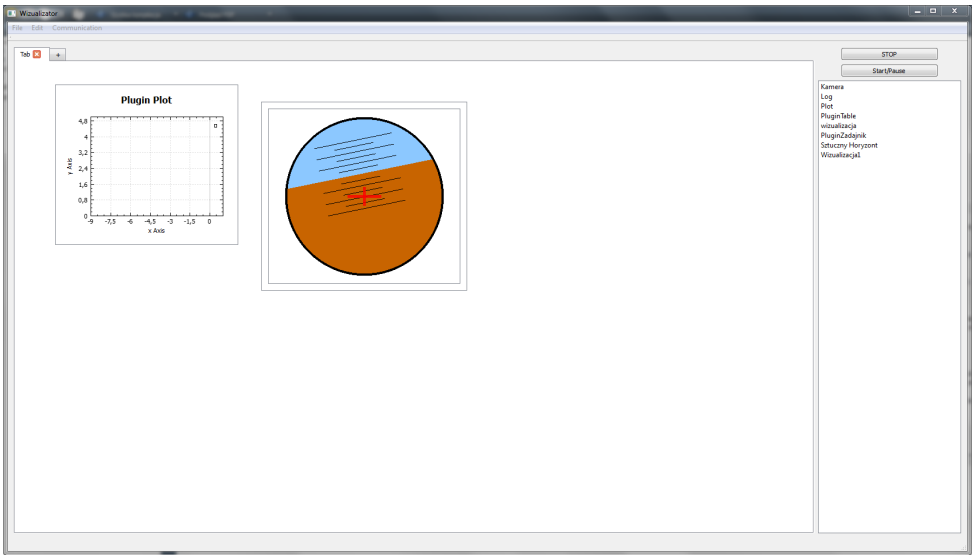
2 Zadania

2.1 Interfejs diagnostyczny

2.1.1 Wygląd i funkcjonalność interfejsu

Na rysunku przedstawiono wygląd głównego okna aplikacji, składa się ono z:

- paska menu,
- okna zakładek,



Rysunek 2: Wygląd głównego okna aplikacji

- okna dostępnych pluginów,
- przycisków obsługujących wykonywanie skryptu,

w kolejnych paragrafach zostanie dokładniej omówiona funkcjonalność poszczególnych elementów.

Pasek menu Na pasku znajdują się trzy rozwijane menu z następującymi opcjami:

- **File**
  - **New interface** – usuwa wszystkie zakładki i widgety, jeżeli stary interfejs nie jest zapisany, aplikacja pyta użytkownika czy chce go zapisać.
  - **Save interface** – zapisuje bieżące ustawienia interfejsu do wybranego pliku.
  - **Load interface** – wczytuje ustawienia interfejsu z wybranego pliku, jeżeli stare ustawienia nie są zapisane aplikacja pyta użytkownika czy chce go zapisać.
  - **Close** – pyta użytkownika czy chce zapisać ustawienia a następnie zamyka aplikację.
- **Edit**
  - **Undo** – cofa ostatnią akcję edycji interfejsu, aplikacja przechowuje do 10 ostatnich zmian.
  - **Redo** – przywraca ostatnią cofniętą akcję zmiany interfejsu, przechowuje do 10 zmian. Przy dokonaniu zmian w interfejsie bufor przechowujący dane jest zerowany.

  Przyciski te są wyszarzane gdy cofnięcie/przywrócenie nie jest możliwe.
- **Communication**
  - **Connect** – nawiązuje połączenie z robotem, przycisk jest wyszarzany gdy połączenie jest już nawiązane.

  - **Disconnect** – przerywa połączenie, przycisk jest wyszarzany gdy połączenie jest nieaktywne.
  - **Configuration** – konfiguracja połączenia, pozwala na ustalenie IP i portu robota, IP komputera na którym jest aplikacja oraz częstotliwości z jaką będą odbierane dane.
  - **Send script** – wysyła skrypt z algorytmem robota.

Zarządzanie zakładkami Nowe zakładki można dodawać poprzez wybranie przycisku ze znakiem + znajdującym się za ostatnią zakładką. Po kliknięciu prawym przyciskiem myszy(PPM) na zakładkę otwiera się menu opcji dostępnych dla zakładki, są to:

- **Change name** – pozwala na zmianę nazwy zakładki. Nazwę można również zmieniać poprzez podwójne kliknięcie lewym przyciskiem myszy(LPM). Zmianę nazwy należy zatwierdzić klawiszem *ENTER*.
- **Clear tab** – usuwa wszystkie widgety znajdujące się na danej karcie.
- **Close tab** – zamyka wybraną kartę, opcja dostępna również przy kliknięciu ikony zamknięcia lub kliknięcia środkowym przyciskiem myszy(ŚPM) na wybranej zakładce.

Przy zamykaniu lub czyszczeniu zakładki aplikacja pyta użytkownika o potwierdzenie, dając jednocześnie możliwość zaznaczenia opcji nie pytania ponownie.

Zarządzanie widgetami Zarządzanie widgetami wykonywane jest przy pomocy mechanizmu *"Przeciągnij i upuść"*. W celu dodania nowego widgetu wybieramy go z listy i przeciągamy go w wybrane miejsce na wybranej zakładce. Utworzony widget możemy złapać poprzez kliknięcie i przytrzymanie PPM przy lewej lub górnej krawędzi widgetu, w ten sposób możemy przesuwać widget obrębie zakładki. Aby usunąć widget należy go przeciągnąć z powrotem na okno dostępnych pluginów. Rozmiar widgetów można zmieniać poprzez złapanie go za dolną lub prawą krawędź i przeciągnięcie do osiągnięcia wymaganej wielkości.

Przyciski obsługi wykonywania skryptu W aplikacji znajdują się dwa przyciski obsługujące wykonywanie skryptu:

- **STOP** – zatrzymuje wykonywanie skryptu, pełni rolę stopu awaryjnego dla skryptu.
- **Start/Pause** – pozwala na zatrzymanie i wznowienie wykonywania skryptu, a także uruchomienie go od nowa po zastopowaniu.

Wczytywanie konfiguracji domyślnej i zapisywane ustawienia Aplikacja przy starcie wczytuje domyślny plik ustawień *„./settings/default_settings.xml"*, jeśli taki plik nie istnieje tworzony jest czysty interfejs. W plikach konfiguracyjnych zapisywana jest wielkość i położenie okna głównego, konfiguracja połączenia z robotem, zakładki, ich nazwy oraz zawartość – tzn. pluginy, ich wielkość i rozmieszczenie. Zapisywane ustawienia pluginów zależą od ich implementacji.

2.1.2 Zarządzanie pakietami TCP

Zarządca pluginami, w celu uproszczenia współpracy wtyczek z serwerem po protokole TCP/IP, został wyposażony w dodatkową funkcjonalność. Każda wtyczka (plugin) daje znać których pakietów chce używać wywołując odpowiednią funkcję *addPackageName*. Po wykonaniu tych operacji zarządca będzie rozsyłać odpowiednie pakiety tylko tym pluginom, które tego będą potrzebowali. Ze strony wtyczek będzie wywoływana funkcja *packageAvailable*. Dodatkowo pluginy będą posiadali możliwość jednorazowego otrzymania pakietu na żądanie. Żeby wysłać żądanie na pobranie pakietu plugin powinien wywołać funkcję *getPackage*. Żądany pakiet przyjdzie w wywołaniu funkcji *packageAvailable*.

2.2 Pluginy

2.2.1 Plugin do rysowania wykresów

Aplikacja do akwizycji danych powinna mieć możliwość graficznej reprezentacji pomiarów. Wyniki przedstawione w formie wykresu są czytelne i jeśli są aktualizowane w czasie rzeczywistym pozwalają na kontrolę urządzenia. Zgodnie z tym założeniem aplikacja powinna zawierać *Plugin* służący do rysowania wykresów.

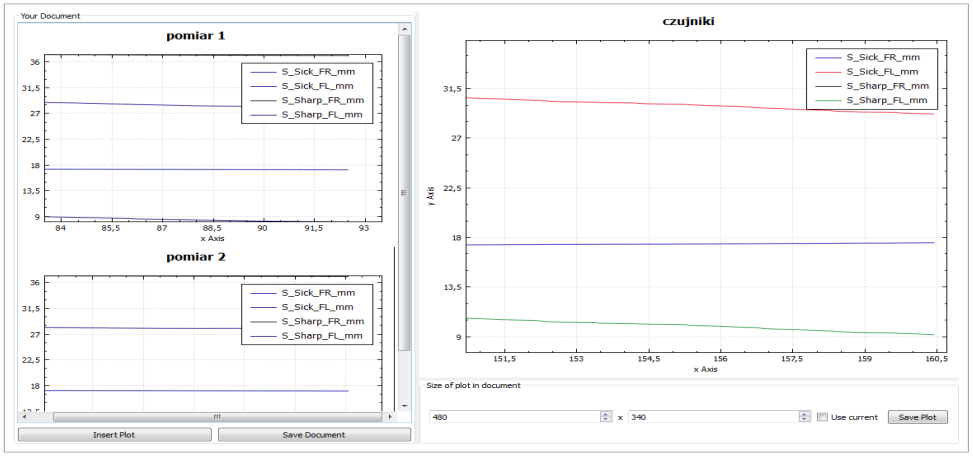
Założenia W celu stworzenia wszechstronnej oraz przyjaznej dla użytkownika aplikacji przyjęto następujące założenia:

- przejrzysty, prosty interfejs,
- możliwość wizualizowania dowolnych parametrów robota,
- automatyczne skalowanie wykresów w zależności od danych,
- opcja zatrzymywania odświeżania wyników,
- możliwość wykonywania zrzutów ekranów.

Realizacja Pluginu Wtyczkę oparto o bibliotekę `qcustomplot` dostępną pod adresem `http://www.qcustomplot.com/`. Autorzy biblioteki dostarczają szeregu przykładów, które w prosty sposób można było zaadaptować do pisanej aplikacji. Wtyczka jest przygotowana i zgodna z interfejsem *Pluginów* opisanych w rozdziale 2.1.

Przepływ informacji Dane, które mogą być wyświetlane na wykresach są dostępne w pliku konfiguracyjnym. Z punktu widzenia użytkownika oznacza to, że w przyszłości w pluginie będzie można w prosty sposób dodawać możliwe do wyświetlenia dane (modyfikacja pliku konfiguracyjnego).

Interfejs użytkownika Główne okno aplikacji zostało przedstawione na rysunku 3. W jej centralnym punkcie znajduje się `widget` związany z wyświetlaniem wykresów.



Rysunek 3: Wygląd Plugina związanego z rysowaniem wykresów

Menu Interfejs oparto o dynamicznie rozwijane menu kontekstowe, które są zależne od wyboru użytkownika. Kliknięcie prawym przyciskiem myszy powoduje pojawienie się odpowiednich menu.

Rysunek 4(a) przedstawia główne menu kontekstowe natomiast jego opcje to:

- Set Plots — Pozwala na wybór danych, które mają być aktualnie wyświetlane na wykresie. Wybór tej opcji wywołuje okno dialogowe przedstawione na 5(f). W łatwy sposób można wybierać zgrupowane sygnały poprzez kliknięcie odpowiedniego `checkboxa`.

- Show Legend - Opcja jest normalnie ukryta. Staje się aktywna, gdy legenda jest schowana i pozwala na przywrócenie legendy.

- Set Color — Opcja jest normalnie ukryta. Staje się aktywna gdy jest zaznaczony dowolny wykres. Opcja pozwala na ustawienie koloru wykresy poprzez okno dialogowe przedstawione na rysunku 5(c).

- Save Plot — Opcja pozwala zapisać wykres. Jej zaznaczenie powoduje pokazanie się dodatkowych opcji pozwalających na ustawienie rozmiaru zapisywanego wykresu. Interfejs przedstawiono na rysunku 6(b). Wciśnięcie przycisku(pojawia się gdy opcja jest aktywna) powoduje pojawienie się okna dialogowego 5(e) umożliwiającego zapis wykresy do pliku w formatach `*.png,*.bmp,*.jpg,*.pdf`.

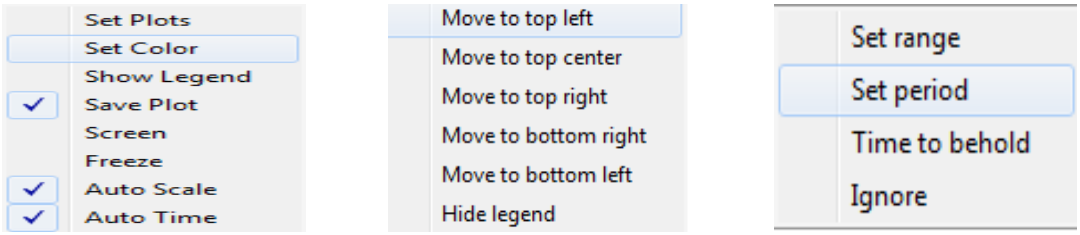
- Screen — Opcja również służy do dokumentowania pomiarów. Jej uaktywnienie powoduje rozszerzenie `Plugina` o plansze, do której można dodawać wykresy oraz je komentować. Interfejs przedstawiono na rysunku 6(a).

- Freeze — Pozwala na zastopowanie dodawania danych do wykresu. Opcja dodatkowo wyłącza autoskalowanie oraz automatyczne podążanie za czasem.

- Auto Scale — Opcja pozwala na dopasowanie skali do aktualnie znajdujących się wykresów.

- Auto Time — Akcja wyzwala śledzenie danych z czasem.

Rysunek 4(b) przedstawia menu związane z legendą, w którym można wykonać następujące akcje:



(a) Główne menu (b) menu legendy (c) menu osi

Rysunek 4: Menu kontekstowe dostępne dla użytkownika

- Move to [...] — Opcja pozwala zmienić położenie legendy w zależności od wyboru miejsca ([...]).
- Hide legend — Akcja ukrywa legendę, którą można ponownie przywołać z głównego menu.

Rysunek 4(c) przedstawia menu związane z osiami, w których można ustawić następujące parametry:

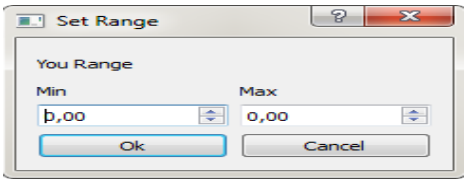
- Set Range — Służy do ustawiania zakresu osi. Wywoływane jest okno dialogowe 5(a)
- Set Period — Opcja dostępna tylko przy osi x, określa czas jaki jest reprezentowany na wykresie przy włączeniu opcji `auto Time`(główne menu kontekstowe).
- Time to behold — Opcja dostępna tylko przy osi x, pozwala na ustawienie horyzontu czasowego przetrzymywanych danych.
- Ignore — Opcja dostępna tylko przy osi x, pozwala na ustawienie ignorowania części danych. Przykładowo gdy wartość jest ustawiona na 4 oznacza to, że co 4 próbka będzie dodawana do wykresu.

Na rysunku 5 przedstawione są dodatkowe okna dialogowe związane z plaginem. Ich działanie jest intuicyjne. Ich funkcjonalność opisano wraz opcjami menu.

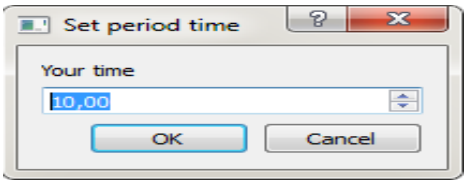
Zmiana opisów Użytkownik ma możliwość zmiany opisów osi, poszczególnych wykresów oraz głównego tytułu. Podwójne kliknięcie w dowolny opis powoduje pojawienie się okna dialogowego przedstawionego na rysunku 5(d). Zmiana opisów jest szczególnie użyteczna przy tworzeniu dokumentacji pomiarów. Należy zaznaczyć, że zmiana nazwy wykresów jest lokalna i służy wyłącznie użytkownikowi. W oknie dialogowym **??** nazwy pozostaną niezmienione.

Skalowanie Wykresy można skalować w osi x lub y, żeby efekt był widoczny, odpowiednia opcja skalowania związana z osiami musi być nieaktywna (opis menu głównego). Możliwe są trzy tryby skalowania:

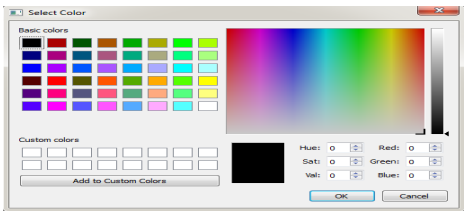
- z zaznaczoną osią x,
- z zaznaczoną osią y,
- bez oznaczania osi.



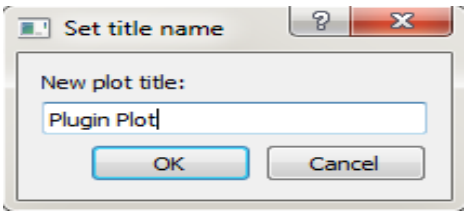
(a) Wczytywanie zakresów



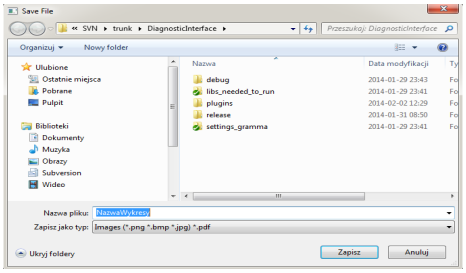
(b) Ustawianie paramentów



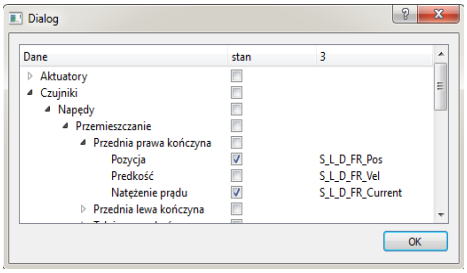
(c) Ustawianie koloru



(d) Zmiana opisów



(e) Zapisanie wykresów



(f) Wybór wykresów

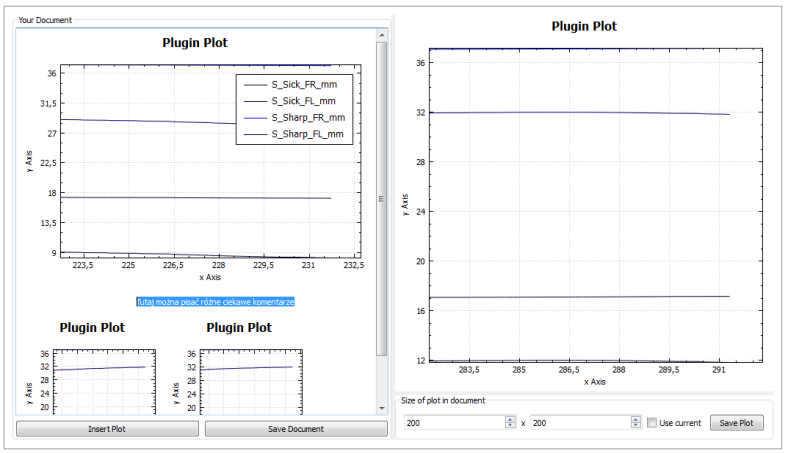
Rysunek 5: Okna dialogowe

Wybranie pierwszej lub drugiej opcji jest realizowane poprzez kliknięcie na odpowiednią oś. Staje się ona wówczas niebieska. Jednocześnie zostaje dezaktywowana odpowiednia opcja automatycznego skalowania. Użycie kołka myszki powoduje zmianę zakresu. Natomiast kliknięcie lewym przyciskiem i przesuwanie myszy pozwala na przesuwanie wykresu wzdłuż zadanej osi. Trzecia opcja skalowania działa analogicznie, ale w dwóch osiach.

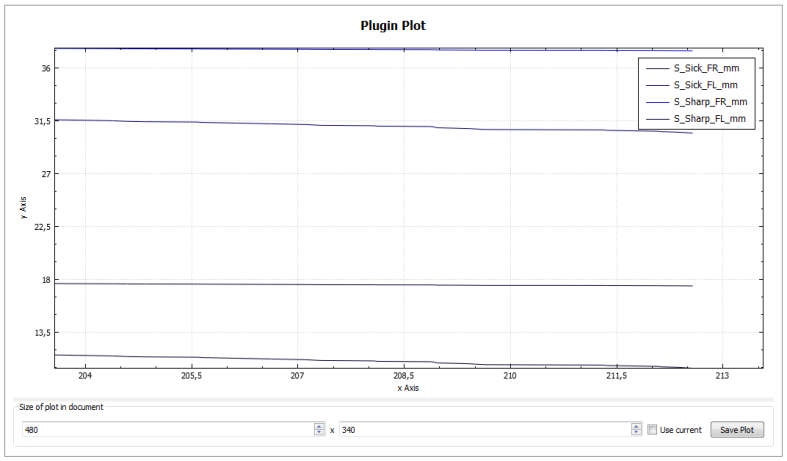
Zapisywanie wykresów Dokumentacje wykresów można wykonywać na dwa sposoby. Wcześniej opisana opcja menu głównego `Screen` rozszerza widget o pole, w którym można wpisywać komentarze oraz dodawać aktualny stan wykresu (wykres 6(a)). Cały dokument można później zapisać w formacie `*.pdf`. Kolejną opcją jest `Save plot`, który umożliwia dokumentowanie wykresów bez podglądu oraz zapisywane do formatów:

- `*.png`,
- `*.bmp`,
- `*.jpg`,
- `*.pdf`.

Podobnie jak wcześniej rozszerza się widget o opcje do ustawiania rozmiaru zapisywanego obrazu (rysunek 6(b)).



(a) Zapisywanie wykresów z wykorzystaniem dokumentowania



(b) Zapisywanie aktualnego wykresu bez podglądu

Rysunek 6: Dokumentowanie danych

2.2.2 Plugin Zadajnik

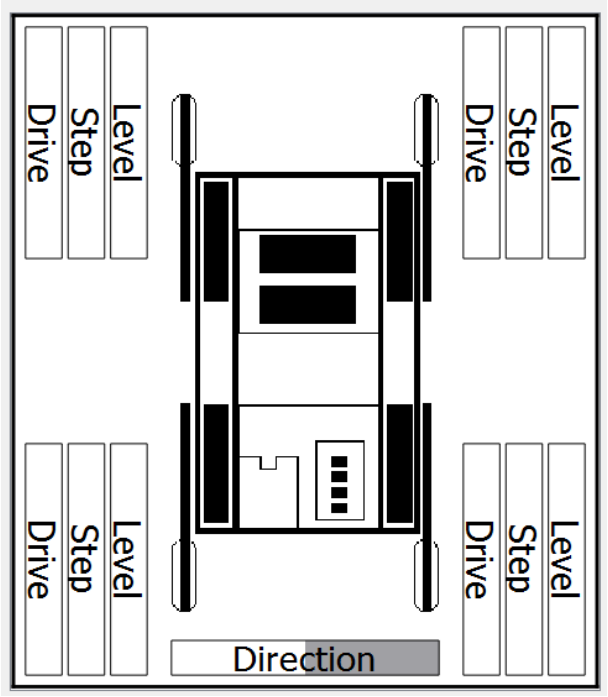
Plugin służy do obserwowania najważniejszych parametrów ruchu robota. Tego typu wizualizacja pozwala na szybką weryfikację stanu. Aplikacja może być szczególnie użyteczna przy sprawdzeniu czy w wyniku sterowania uzyskiwane są wartości zadane.

Przepływ informacji W pluginie przedstawione są następujące sygnały

- prędkość `Drive`,
- położenie `Step`,
- położenie `Level`,
- kierunek jazdy `Direction`.

Pierwsze trzy parametry są przedstawione dla każdego z kół.

Interfejs użytkownika Użytkownik nie bezpośredniego wpływu na działanie aplikacji. Poszczególne parametry są wyświetlane w formie słupków o zmiennym wypełnieniu. Jeżeli parametr ma wartość 0 oznacza to pusty słupek. Wartość ujemna powoduje wypełnienie słupka w dół bądź na lewo w przypadku `Direction`. Na rysunku 7 przedstawiono wygląd aplikacji.



Rysunek 7: Wygląd Plugina pozwalającego na zobrazowanie podstawowych parametrów robota

2.2.3 Plugin Tablice

Graficzna reprezentacja wyników może być pewnych sytuacjach nie wystarczająca. Szczególnie wtedy, gdy pojawia się potrzeba dokładnej znajomości parametru lub grupy parametrów. Zgodnie z tym założeniem pojawia się potrzeba stworzenia aplikacji reprezentującej dane w formie numerycznej. Zdecydowano się na plugin, który w formie tabeli przedstawia poszczególne dane.

Przepływ informacji Dane, które mogą być wyświetlane w tabeli są dostępne w pliku konfiguracyjnym. Z punktu widzenia użytkownika oznacza to, że w przyszłości w pluginie będzie można w prosty sposób dodawać możliwe do wyświetlenia dane (modyfikacja pliku konfiguracyjnego).

Interfejs użytkownika Rysunek 8 przedstawia wygląd aplikacji. W pierwszej kolumnie jest przedstawiony aktualny czas. Kolejne kolumny reprezentują poszczególne parametry robota. Użytkownik ma możliwość zmiany obserwowanych parametrów poprzez menu kontekstowe.

	time	Sharp_FR_m	Sharp_FL_m	Sick_FR_mr	Sick_FL_mr	S_F_FR_N	S_F_FL_N	S_F_RR_N	S_F_RL_N	S_Accel_X	S_Accel_Y	S_Accel_Z
1	12:44:18.128	41.8532	8.49265	21.8532	28.4927	14.2239	29.7882	0	0	-0.429759	4.03702	5.65725
2	12:44:18.047	41.8611	8.46667	21.8611	28.4667	14.3647	29.7833	0	0	-0.337468	4.06164	5.64205
3	12:44:17.286	41.8689	8.44129	21.8689	28.4413	14.5307	29.7784	0	0	-0.241805	4.09226	5.62659
4	12:44:17.085	41.8767	8.41654	21.8767	28.4165	14.7155	29.7736	0	0	-0.143726	4.12869	5.61087
5	12:44:17.076	41.8845	8.39243	21.8845	28.3924	14.9117	29.7687	0	0	-0.0442112	4.1707	5.59492

Rysunek 8: Wygląd Plugina pozwalającego na zobrazowanie podstawowych parametrów robota

Menu Menu stanowi interfejs użytkownika. Możliwe opcje pojawiają się przy kliknięciu prawym przyciskiem myszy na obszarze tablic. Dostępne akcje przedstawiono poniżej:

- Set Data — Podobnie jak w przypadku plugina Plot uruchamia okno dialogowe 5(f), które pozwala na wybór sygnałów, które mają być reprezentowane w tablicy.
- Rows remember — Ustawia horyzont pamiętanych danych (ilość wierszy).
- Columns width — Pozwala na ustawienie szerokości kolumn.

2.2.4 Plugin Kamery

W aplikacji zapewniono możliwość podglądu obrazu z kamery na robocie. Umożliwi to zdalną ocenę sytuacji oraz rozwiązanie problemów robota. Kamera powinna zostać podpięta pod port USB komputera z Interfejsem diagnostycznym.

Założenia Aby zapewnić funkcjonalność oraz możliwość rozwoju aplikacji przyjęto:

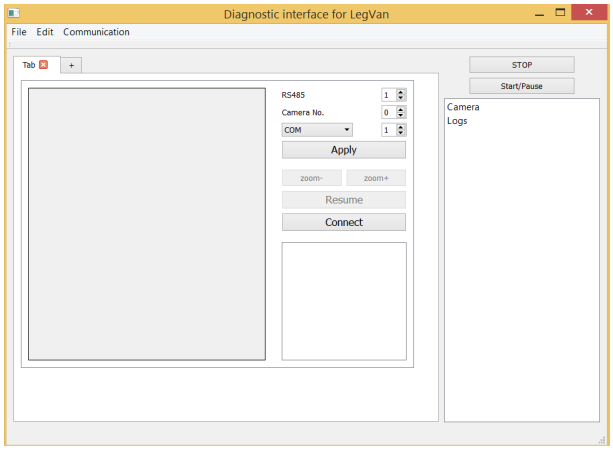
- przejrzysty, prosty interfejs,
- możliwość wyboru kamery,

- automatyczne skalowanie obrazu,
- możliwość zatrzymywania obrazu,
- możliwość zoomu optycznego z kamery (protokół Pelco-D),
- możliwość wyboru portu RS485 pod który wpięta jest kamera, przy komunikacji w protokole Pelco-D,
- możliwość wyboru portu RS232 pod którym wpięta jest kamera na komputerze robota.

Realizacja Pluginu Wtyczkę oparto o bibliotekę OpenCV dla języka C++. Wtyczka jest przygotowana i zgodna z szablonem dla *Pluginów* opisanych w rozdziale 2.1.

Przepływ informacji Przy włączeniu plugina należy wybrać kamerę oraz skonfigurować jej ustawienia. Po połączeniu, zostaje podłączona kamera o zadanym adresie. Można zmienić adres wyświetanej kamery podczas działania programu. Należy wówczas ponownie połączyć się z kamerą. Wtyczka nie pobiera żadnych informacji od aplikacji, wysyła jedynie komunikaty protokołu Pelco-D, które następnie przesyłane są przez aplikacje do robota na którym jest kamera.

Interfejs użytkownika Interfejs użytkownika został przedstawiony poniżej.



Rysunek 9: Wygląd Plugina wyświetlającego obraz z kamery

Kompilacja Pluginu Do kompilacji pluginu potrzebna jest biblioteka OpenCV dla języka C++ w wersji 2.3.1 lub nowszej kompatybilnej.

2.2.5 Plugin Logi

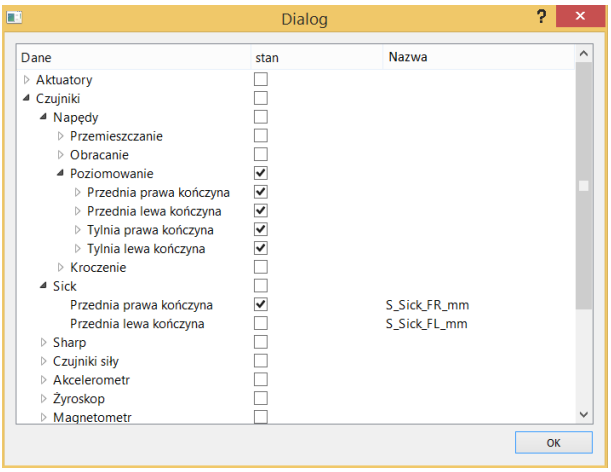
Aplikacja powinna mieć możliwość zapisu zgromadzonych informacji na dysku. Zapewnia jej to aplikacja logów, w której istnieje możliwość zapisu wszystkich informacji wysłanych przez robota. Dane nie są poddane żadnej obróbce, dzięki czemu można szybko zdiagnozować przyczynę powstania błędu w komunikacji lub zaimplementowanych algorytmach.

Założenia Przyjęto następujące założenia plugina:

- zapis danych do formatów: txt, csv, csv(Excel),
- standardowo dane są zapisane pod nazwą „log_dd_MM_yyyy_hh_mm_s”,
- istnieje możliwość zmiany nazwy,
- istnieje możliwość wybrania danych potrzebnych do zapisu.

Realizacja pluginu Do realizacji pluginu użyto standardowych bibliotek Qt. Dodatkowo użyto specjalnego widgetu, umożliwiającego prosty wybór danych do zapisu. Wtyczka jest zgodna z szablonem dla *Pluginów* opisanych w rozdziale 2.1.

Widget Widget umożliwiający prosty wybór danych, został napisany uniwersalnie do użytku w wielu pluginach. Jest prosty w użyciu, posiada możliwość zaznaczenia i odznaczenia wielu zmiennych naraz. Po zaznaczeniu zmiennej, ta automatycznie ustawia się jako aktywna.



Rysunek 10: Wygląd widgetu z wyborem danych

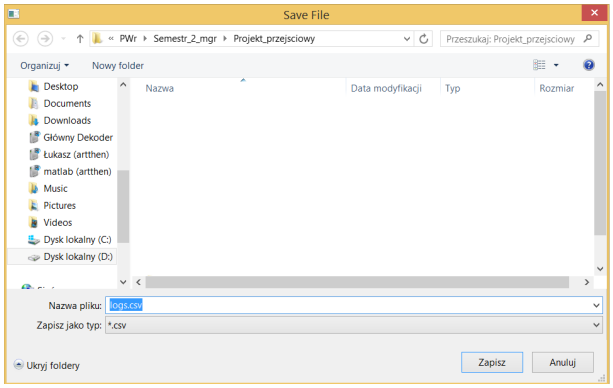
Interfejs użytkownika Użytkownik ma możliwości zawarte w założeniach, czyli:

- wybór formatu zapisu danych,
- wybór danych potrzebnych do zapisu,
- utworzenie nazwy pliku i wybór miejsca zapisu.

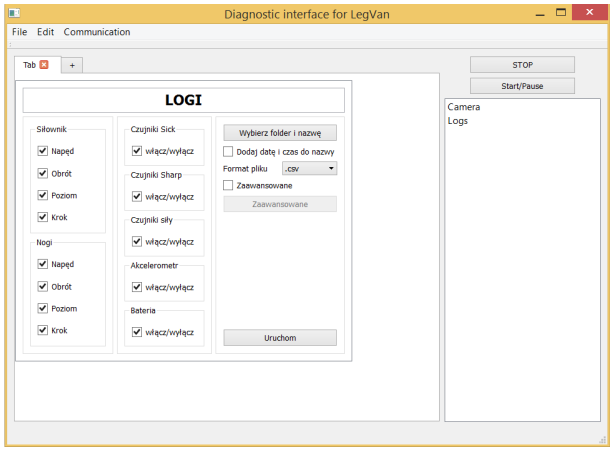
Format „.csv(Excel)” powstał dla systemów operacyjnych (Windows) z separatorami listy ustawionymi na znak średnika „ ; ”. Pakiet Office korzysta z ustawień regionalnych. W innych pakietach (np. LibreOffice) działa podstawowy format „.csv”.

2.2.6 Plugin wizualizacji robota

Dla dobrej diagnostyki robota napisano wtyczkę wizualizującą robota. Jest to rzut z góry na robota.



Rysunek 11: Wygląd okna zapisu danych



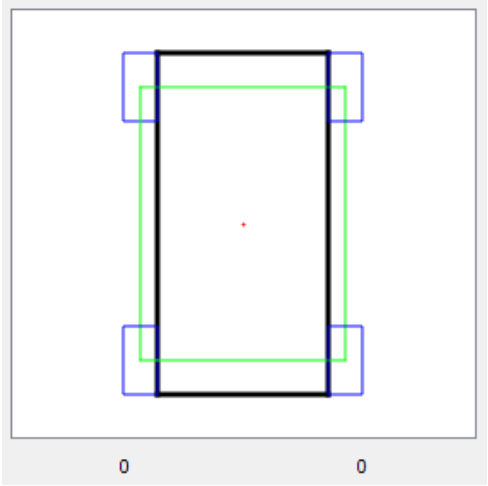
Rysunek 12: Wygląd Plugina Logów

Realizacja Pluginu Wtyczka realizuje graficzne przedstawienie robota. Jest to rzut z góry z zaznaczonym korpusem i kołami robota. Na tle robota jest odrysowywany wielokąt podparcia i środek masy wyliczany na podstawie odczytów z czujników siły umieszczonych na każdej z kończyn. Wtyczka jest przygotowana i zgodna z szablonem dla *Pluginów* opisanych w rozdziale 2.1.

Przepływ informacji Wtyczka pobiera potrzebne informacje:

- odczyt z czujników siły z kończyn,
- odczyt w położenia w przegubach we wszystkich kończynach.

Rysowanie pozycji kół i wielokąta podparcia Dla normalnej pozycji robota (wszystkie koła mają styczność z podłożem) koła odrysowywane są na niebiesko, wielokąt podparcia jest połączeniem środków kół i jest odrysowywany w kolorze niebieskim. Środek ciężkości zaznaczony jest czerwonym punktem, a jego współrzędne względem środka geometrycznego robota wypisywane są pod wizualizacją. Omówioną sytuację pokazano na rysunku 13.



Rysunek 13: Widok robota w pozycji początkowej.

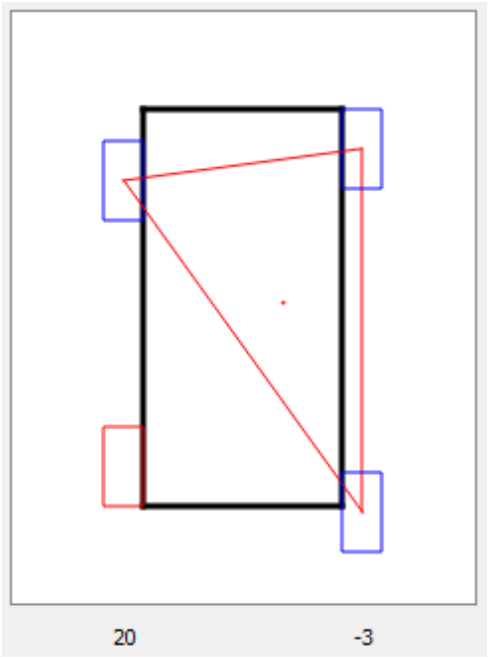
Kolory zmieniają się w sytuacji gdy któraś z kończyn robota traci kontakt z podłożem. Wtedy koła nie mające styczności z ziemią i zarówno wielokąt podparcia są odrysowywane na czerwono. Przykładową konfigurację przedstawiono na rysunku 17.

Wyliczanie pozycji kół Dla prawidłowej wizualizacji należy obliczyć rzut pozycji koła na płaszczyznę podłoża równoległą do robota. Posiadamy jedynie odczyty z siłowników q_1 i q_2. Skorzystano z wyliczeń dokonanych w pracy doktorskiej dr Jarosława Szreka w rozdziale 3.1 Model Kinematyki.

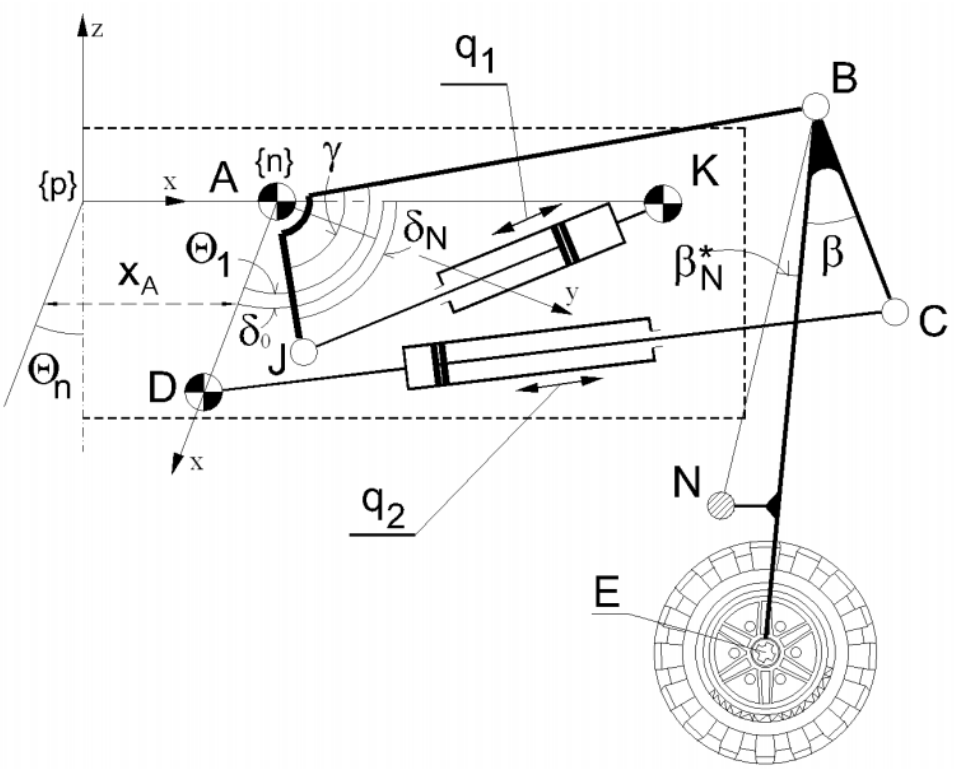
Oznaczenia przyjęte w pracy doktorskiej przedstawiono na rysunku 15.

Współrzędne punktu A to (0,0). Do wyliczenia współrzędnych punktu B korzystamy z zależności:

$$x_B = l_{AB} \cos\theta_1, \;\; y_B = l_{AB} \sin\theta_1.$$



Rysunek 14: Widok robota z jedną kończyną podniesioną.



Rysunek 15: Przyjęte oznaczenia.

Następnie wyliczamy współrzędne punktu C:

$$x_C = C_1 - \frac{2y_B y_C}{2(x_B - l_{AD})},$$
$$C_1 = \frac{l_{AB}^2 - l_{BC}^2 + q_2^2 - l_{AD}^2}{2(x_B - l_{AD})},$$
$$y_C = \frac{-s + \sqrt{s^2 - 4pw}}{2p},$$

gdzie:

$$p = \frac{y_B^2}{(x_B - l_{AD})^2} + 1,\ w = (l_{AD} - C_1)^2 - q_2^2,\ s = \frac{2y_B(l_{AD} - C_1)}{x_B - l_{AD}}.$$

Potrzebny kąt θ_1:

$$\theta_1 = \gamma + \delta_0 - \delta_N$$
$$\delta_N = \arccos(\frac{l_{AJ}^2 + l_{AK}^2 - q_1^2}{2l_{AJ}l_{AK}})$$

By wyliczyć θ_2 korzystamy:

$$\theta_2 = \arctan 2(y_C - y_B, x_C - x_B)$$

Wyliczenie współrzędnej x końca naszej kończyny otrzymujemy z:

$$x = l_{AB}\cos\theta_1 + l_{BE}\cos(\theta_2 - \beta).$$

W kodzie przyjęto takie same oznaczenia, by umożliwić ewentualne zmiany w geometrii robota.

2.2.7 Plugin do odczytu danych z sensora Kinect

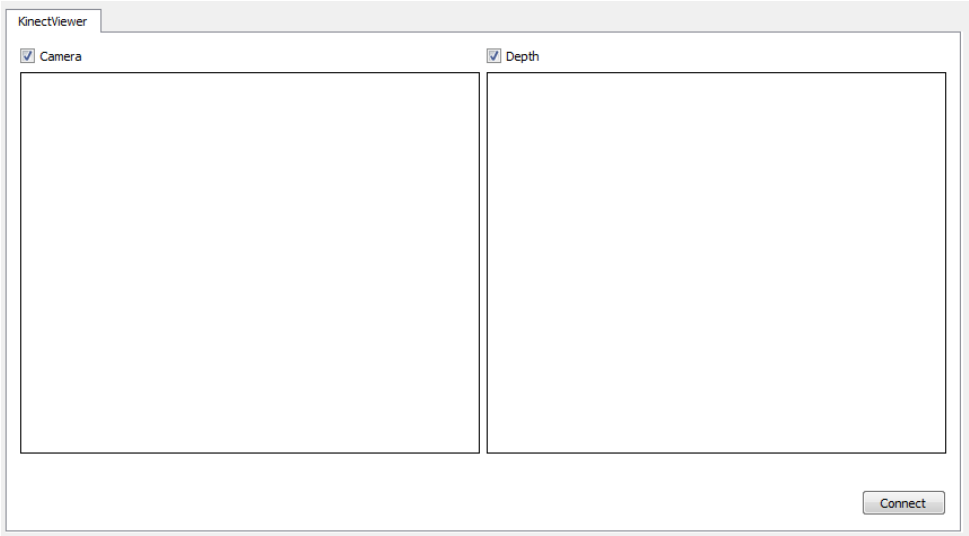
Ponieważ istnieje możliwość wyposażenia robota w sensor Kinect zdecydowano o napisaniu wtyczki do aplikacji diagnostycznej pozwalającej łączyć się i wyświetlać dane zbierane przez ten sensor.

Realizacja Pluginu Wtyczkę oparto o bibliotekę QKinectWrapper. Plugin oparto o przykłady dostarczone przez autorów biblioteki. Wtyczka jest przygotowana i zgodna z szablonem dla *Pluginów* opisanych w rozdziale 2.1.

Przepływ informacji Wtyczka nie pobiera żadnych danych poprzez komputer robota. Bezpośrednio łączy się z Kinectem i wyświetla otrzymane dane.

Interfejs użytkownika Rozpoczęcie pracy rozpoczyna się poprzez naciśniecie przycisku "Połącz". Pod okienkami wyświetlającymi bieżący obraz, znajduje się informacja o bieżącym stanie sensora. Jeżeli połączono się z Kinectem i jest wyświetlany obraz, otrzymujemy także informację o prędkości z jaka pobierany jest obraz, a także która klatka jest aktualnie wyświetlana.

Użytkownik ma możliwość rozłączenia się z urządzeniem, a także zamrożenia każdego z obrazów z osobna: z kamery i mapy głębi.



Rysunek 16: Wygląd interfejsu użytkownika.

Kompilacja Pluginu Do skompilowania wtyczki potrzebne są zasoby, z których korzysta biblioteka QtKinectWrapper i jednocześnie Kinect:

- OpenNI (biblioteka powinna być umieszczona na poziomie katalogu z źródłem wtyczki),
- PrimeSense for Kinect,
- NITE.

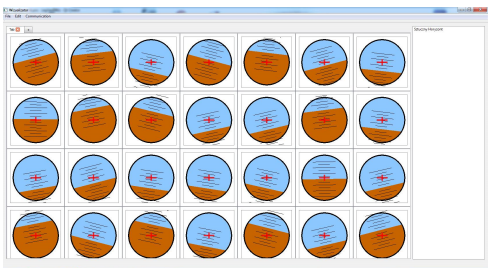
2.2.8 Plugin Sztuczny Horyzont

Dla dobrej diagnostyki robota napisano wtyczkę wizualizującą położenie robota względem poziomu.

Realizacja Pluginu Wtyczka realizuje graficzne przedstawienie położenia robota w stosunku do poziomej powierzchni (tak jak ma to miejsce w samolotach). Wtyczka jest przygotowana i zgodna z szablonem dla *Pluginów* opisanych w rozdziale 2.1.

Przepływ informacji Wtyczka pobiera potrzebne informacje:

- odczyt z akcelerometru.



Rysunek 17: Wizualizacja położenia robota w różnych momentach

2.3 Komendy

Implementacja komend pozwoli użytkownikowi na łatwe wydawanie poleceń dla robota z poziomu skryptu bez konieczności pisania i kompilowania całości programu z algorytmem sterowania. Jest to istotna zaleta dla platformy LegVanII, ponieważ jednym z celów dla którego powstala jest testowanie algorytmów starowania robotem kroczącym.

2.3.1 Low Level Commands

Komendy niskiego poziomu działają bezpośrednio na strurze danych robota. Przypisują zadane parametry w odpowiednie miejsca oraz zwracają na żądanie odczyty z czujników robota.

- void UCmd::SetP(int motorID, int reg, double value)

  Ustawia parametr P w skrukturze robota dla wybranej regulacji

  - motorID - ID silnika - (0 - Drive), (1 - Turn), (2 - Level), (3 - Step)
  - reg - Typ regulacji - (0 - Pos), (1 - Vel)
  - value - Wartość parametru P

- void UCmd::SetI(int motorID, int reg, double value)

  Ustawia parametr I w skrukturze robota dla wybranej regulacji

- void UCmd::SetD(int motorID, int reg, double value)

  Ustawia parametr D w skrukturze robota dla wybranej regulacji

- void UCmd::SetReg(int motorID, int type)

  Ustawia tryb w jakim sterowany jest robot

  - motorID - ID silnika - (0 - Drive), (1 - Turn), (2 - Level), (3 - Step)
  - type - Tryb regulacji - (0 - PWM), (1 - Pos), (2 - Vel)

- void UCmd::SetVel(int motorID, int legID, double value)

  Ustawia wartość predkości

  - motorID - ID silnika - (0 - Drive), (1 - Turn), (2 - Level), (3 - Step)
  - legID - ID nogi - (0 - FL), (1 - FR), (2 - RR), (3 - RL)
  - value - Wartość parametru prędkości

- void UCmd::SetPos(int motorID, int legID, double value)

  Ustawia wartość pozycji

- double UCmd::GetPos(int motorID, int legID)

  Zwraca wartość pozycji

  in motorID - ID silnika - (0 - Drive), (1 - Turn), (2 - Level), (3 - Step)
  - legID - ID nogi - (0 - FL), (1 - FR), (2 - RR), (3 - RL)

- double UCmd::GetVel(int motorID, int legID)
  Zwraca wartość prędkości
- double UCmd::GetCur(int motorID, int legID)
  Zwraca wartość prądu
- double UCmd::GetSick(int legID)
  Zwraca wartość z czujnika Sick
  - legID - ID nogi - (0 - FL), (1 - FR)
- double UCmd::GetSharp(int legID)
  Zwraca wartość z czujnika Sharp
- double UCmd::GetForce(int legID)
  Zwraca wartość z czujnika siły
- double UCmd::GetAccel(int axis)
  Zwraca wartość z akcelerometru
  - axis - oś wzdłuż której odczyt. jest przyśpieszenie - (0 - X), (1 - Y), (2 - Z)
- double UCmd::GetVoltage()
  Zwraca wartość napięcia na akumulatorze

2.3.2 Medium Level Commands

- double UCmd::Turn(int promien, int strona)

Ustawia promień skrętu kół.

- double UCmd::Level(var z)

Ustawia wartość poziomu robota.

- void UCmd::stop()

Zatrzymuje robota.

- double UCmd::Move(int kierunek, int predkosc)

Robot jedzie prosto lub do tyłu.

2.3.3 High Level Commands

Pojawią się w przyszłości.

2.4 Struktura danych

Struktura danych robota została podzielona na dwie części: dane odczytywane z robota i dane wysyłane do robota. Uporządkowanie wszystkich zmiennych w *URBI* miało na celu łatwiejszy dostęp do struktury danych z poziomu każdego modułu *URBI* jak również z poziomu skryptu w języku *urbiscript*.

2.4.1 Dane odczytywane z robota

Dane odczytywane z robota zostały zebrane w strukturę *Sensors*. Dostępne są w niej dane z wszystkich czujników, sprzężenie zwrotne z silników oraz wyjścia ogólnego przeznaczenia. Szczegółowe dane na temat elementów struktury znajduje się w pliku *RobotCore.pdf*.

2.4.2 Dane wysyłane do robota

Dane wysyłane do robota przechowywane są w strukturze *Actuators*. Elementami struktury są wartości zadane na silniki. Dodatkowo struktura umożliwia wybór sposobu sterowania danym napędem oraz jego nastawy regulacji. Szczegółowe dane znajdują się w pliku *RobotCore.pdf*.

2.4.3 Konwersja danych z czujników na dane w jednostkach rzeczywistych

Wartości odczytywane i wysyłane do robota wymagają konwersji na dane o interpretacji fizycznej. Do tego celu służy moduł UDataConverter. Jego zadaniem jest nie tylko konwersja, ale również separacja między odczytywaniem/wysyłaniem surowych danych z robota a korzystaniem z tych danych na poziomie algorytmów sterowania.

2.5 Ramka danych

Aby zapewnić możliwie łatwą wymianę zadajnika, bez konieczności wymiany całej aplikacji powstała koncepcja wirtualnej ramki danych. Ramka ta ma za zadanie przechowywać zunifikowane wartości pochodzące z zadajnika, które następnie zostaną odpowiednio zinterpretowane na komendy dla robota.

Dzięki zastosowaniu wirtualnej ramki danych wymiana zadajnika wiązać się będzie jedynie z odpowiednią unifikacją danych wejściowych.

2.5.1 Pola ramki

1. emergencyStop
   Stop awaryjny w zależności od przyjmowanych wartości pozwala kontynuować ruch lub zatrzymuje robota.

   - `0` - Nieaktywny. Robot może się poruszać.
   - `1` - Aktywny. Ruch robota zostaje wstrzymany.

2. recalibration
   Rekalibracja w zależności od przyjmowanych wartości pozwala na zatrzymanie robota i ponowną kalibracje.

   - `0` - Nieaktywny. Robot może się poruszać.
   - `1` - Aktywny. Następuje ponowna kalibracja robota .

3. mode
   Tryb w zależności od przyjmowanych wartości określa w jakim trybie porusza się robot.

   Tryb jazda. Pozwala na jazdę do przodu i tyłu po zadanej krzywiźnie. Dostępny jest z zawsze, niezależnie od pozostałych trybów.

- `0` - Poziomowanie. Powala na podnoszenie i opuszczanie całego robota.
- `1` - Pochylanie. Pozwala na pochylanie roboto do przodu, tyłu i na boki.
- `2` - Kroczenie. Pozwala na sterowanie każdą nogą z osobna.

4. velocity
   Prędkość w zależności od przyjmowanych wartości określa z jaką prędkością i w jakim kierunku porusza się robot.

   - `0-4095` - Przedział zawiera wartości odpowiadające prędkościom od maksymalnej prędkości w tył do maksymalnej prędkości do przodu.

5. curvature
   Krzywizna w zależności od przyjmowanych wartości określa wartość skręcenia kół robota.

   - `0-4095` - Przedział zawiera wartości odpowiadające skręceniu kół od maksymalnego wychylenia w prawo do maksymalnego wychylenia w lewo.

6. joinID
   ID członu w zależności od przyjmowanych wartości określa którą kończyną poruszamy. Wartość wyliczamy prze konwersje cztero elementowej liczby binarnej na liczbę dziesiętną. Elementy liczby binarnej, począwszy od najmlodszego bitu oznaczają: prawą przednią, lewą przednią, prawą tylną, lewą tylną nogę. Otrzymaną liczbę interpretujemy następująco:

   - `1` - Przednia prawa noga.
   - `2` - Przednia lewa noga.
   - `4` - Tylna prawa noga.
   - `8` - Tylna lewa noga.
   - `3` - Przednie nogi.
   - `12` - Tylnie nogi.
   - `5` - Prawe nogi.
   - `10` - Lewe nogi.

7. rightLeft
   W zależności od przyjmowanych wartości określa prędkość wychylania w prawo lub w lewo. Istotne tylko w trybie Pochylanie.

   - `0-4095` - Przedział zawiera wartości odpowiadające prędkościom od maksymalnej prędkości w tył/lewo do maksymalnej prędkości w przód/prawo.

8. frontRear
   W trybie Pochylanie pole w zależności od przyjmowanych wartości określa prędkość wychylania w przód lub w tył; W trybie Kroczenie pole w zależności od przyjmowanych wartości określa prędkość przesuwania nogi do przodu lub do tył.

   - `0-4095` - Przedział zawiera wartości odpowiadające prędkościom od maksymalnej prędkości w tył do maksymalnej prędkości w przód.

9. upDown
   Pole w zależności od przyjmowanych wartości określa prędkość opuszczania lub podnoszenia nogi. Istotne tylko w trybach Poziomowanie i Kroczenie

   - `0-4095` - Przedział zawiera wartości odpowiadające prędkościom od maksymalnej prędkości w dół do maksymalnej prędkości w górę.

2.5.2 RDS8000

Zadajnikiem sterującym robotem jest aparatura RC RDS8000. Aparatura ta przesyła sygnały do odbiornika 92824Z, który posiada 8 kanałów wyjściowych. Wykonany jest moduł konwersji - sygnał z 8 kanałów jest odczytywany i stosowna zmiana wysyłana jest za pośrednictwem UART'a, a PC widzi port jako zwykły COM.

Za pośrednictwem portu szeregowego odbywa się komunikacja według przyjętego standardu komunikacyjnego [1].

Protokół został opracowany nad miarowo z punktu widzenia obsługi zadajnika, także nie wszystkie przedstawione możliwości komunikacji zostaną wykorzystane.

2.5.3 Identyfikacja aparatury

CH1 - (6) - góra/dół (wartości 050 - 18F - 313)

CH2 - (6) - prawo/lewo (wartości j/w)

CH3 - (5) - prawo/lewo (wartości j/w)

CH4 - (5) - góra/dół (wartości j/w)

CH5 - (3) - on/off (wartości 000-001)

CH6 - (4) - on/off (wartości 000-001)

CH7 - (1) - on/off (wartości 000-001)

CH8 - (2) - dół/środek/góra (wartości 000-001-002)

2.5.4 Sterowanie

CH8 - 0 - emergencyStop = 1
- 1 - recalibration = 1
- 2 - remoteControl
  - CH1 −− > velocity
  - CH2 −− > curvature

CH7 - 0 -
CH5 - 0 - mode = Level
* CH4 −− > upDown
- 1 - mode = Tilt
* CH3 −− > rightLeft
* CH4 −− > frontRear

- 1 - mode = Step
  - CH5 = 0 & CH6 = 0 −− > joinID = Front Right Leg
  - CH5 = 0 & CH6 = 1 −− > joinID = Front Left Leg
  - CH5 = 1 & CH6 = 0 −− > joinID = Rear Right Leg
  - CH5 = 1 & CH6 = 1 −− > joinID = Rear Left Leg
  - CH3 −− > frontRear
  - CH4 −− > upDown

2.5.5 Inicjalizacja

W celu umożliwienia prostej zmiany parametrów zadajnika podstawowe parametry wczytywane są z pliku *config.ini*.

Parametry przyjmowane przez program z pliku są następujące:

1. Parametry zadajnika

   - CH1...4_MIN
     Określa minimalną wartość jaką można odebrać od zadajnika z danego kanału.
   - CH1...4_ZERO
     Określa wartość którą należy interpretować jako neutralną (0).

- CH1...4_MAX
  Określa maksymalną wartość jaką można odebrać od zadajnika z danego kanału.

2. Parametry transmisji

   - PORT - nazwa portu
   - BAUD_ RATE - szybkość transmisji
   - BYTE_SIZE - rozmiar baytu
   - PARITY - parzystość
   - STOP_BITES - bit stopu

3. Inne parametry

   - TIME_DELAY
     Określa dopuszczalną wartość czasu jaki może upłynąć pomiędzy poprawnie odebranymi danymi.

2.5.6 Struktura plików

1. *driver_serial.cpp + driver_serial.h*

   Pliki zawierają klasę `Serial` która dostarcza funkcje pozwalające na połączenie, wysyłanie i odbieranie z portu szeregowego.

   Klasa ta jest autorstwa pana Jana Kędzierskiego. Pobrano z:
   http://http://www.urbiforge.org/index.php/Modules/USerial

2. *rds_8000.cpp + rds_8000.h*

   Pliki zawierają klasę `RDS800` która przechowuje informacje wczytane z pliku *config.ini* oraz dostarcza funkcje pozwalające na konwersje danych odebranych z zadajnika do struktury wirtualnej ramki.

3. *vf2robot.cpp + vf2robot.h*

   Pliki zawierają funkcje pozwalające na konwersje danych z wirtualnej ramki do danych odpowiadających standardom robota oraz odpowiednie przypisanie uzyskanych wyników do struktury robota.

4. *urds8000.cpp + urds8000.h*

   Pliki zawierają klasę `URDS8000` która dostarcza funkcje pozwalające na połączenie, wysyłanie i odbieranie z portu szeregowego z poziomu URBI. Klasa ta także spaja wszystkie powyższe pliki pozwalając na połączenie z zadajnikiem, odbiór danych, konwersje danych do wirtualnej ramki i dalej do robota.

   Klasa `URDS8000` powstała w oparciu o klasę `USerial`, została jednak zancznie rozszerzona i dostosowana na potrzeby aplikacji.

2.5.7 Działanie i użycie

Klasa `URDS8000` dostarcza głównie 2 funkcji pozwalających na rozpoczęcie komunikacji z zadajnikiem i starowania robotem oraz na zakończenie sterowania przez zadajnik.

Funkcja *start()* rozpoczyna procedurę sterowania. Na początku łączy się z portem szeregowym do którego podłączony jest moduł komunikacyjny zadajnika. Jeżeli podłączenie portu powiodło się, to rozpoczyna się wykonywanie w nieskończonej pętli (warunkiem pętli jest połączenie, *connected == 1*) następujących czynności:

1. Wysłanie do modułu żądania danych z kanałów (0xFF 0x11 0x6F 0x10).
2. Sprawdzenie czy odebrane dane z portu tworzą ciąg (0xFF 0x11 0x6) jeśli tak to przechodzimy do punktu 4, jeżeli nie to przechodzimy do punktu 3.
3. Sprawdzamy ile czasu minęło od ostatniego odbioru poprawnej ramki. Jeżeli warunek przerwania komunikacji został spełniony to zatrzymujemy robota i zamykamy połączenie. Jeśli nie został spełniony to wracamy do punktu 1.
4. Odczytujemy 24 znak z portu (po 3 znaki na każdy z 8 kanałów) i konwertujemy je na wirtualną ramkę.
5. Konwertujemy wirtualną ramkę i uaktualniamy strukturę robota. Wracamy do punktu 1.

Funkcja *stop()* kończy procedurę sterowania. Funkcja zatrzymuje robota w aktualnej pozycji i odłącza się od portu.

2.6 Wykrywanie linii

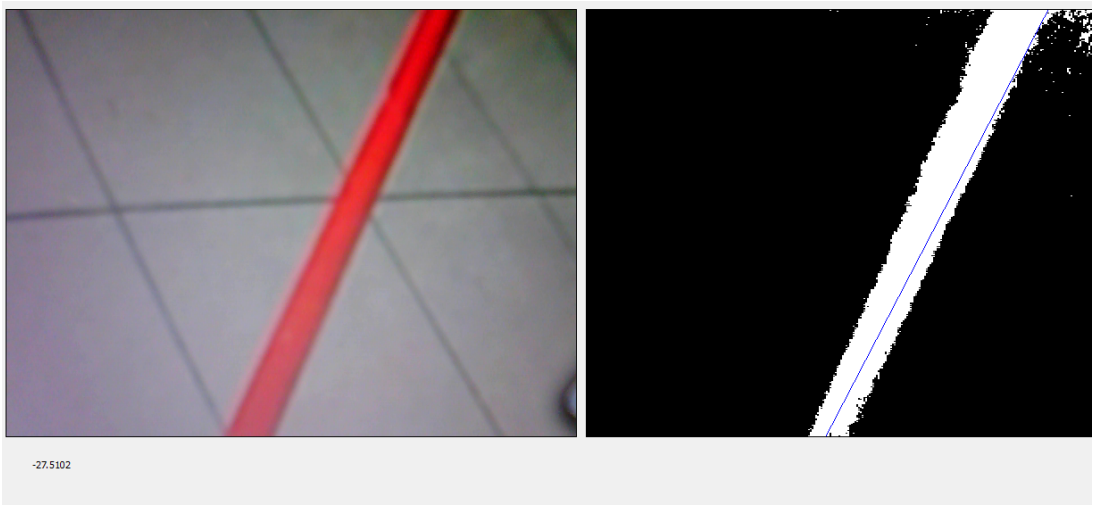
Problem wykrywania linii poprzez obraz z kamery dostepnej na robocie.

2.6.1 Przygotowanie do zadania

Przegląd bibliotek do obsługi grafiki. Większość niestety okazała się komercyjna. Ostatecznie wybrano OpenCV. (wynik testu to 35ms na przetworzenie jednej klatki obrazu)

2.6.2 Algorytm

Zbadano kilka różnych sposobów wykrywania linii. Na początku podjęto próby wykrycia linii czarnej.Okazało się to dość złożonym zadaniem - występowały duże szumy.Próby wykorzystania metod obliczających pochodną obrazu zawodziły, ze względu na złożoność obliczeniową. Ostatecznie wybrano wykrywanie czerwonej linii. Pewne cechy kolorów spowodowały, że wykorzystując proste równania algebraiczne można odfiltrować obiekty o innych kolorach niż określone. Następnym krokiem była próba identyfikaci obiektów na obrazie, niestety zawiodła ona przez złożonść obliczeniową (w pewnych przypadkach jedna klatka obrazu wyliczała się ponad 30sek) Ostatecznie wykorzystano prostą aproksymacje linią, a efektem pracy jest funkcja C++ zwracająca kąt wykrytej linii. Sam algorytm ma spore ograniczenia, niemniej jest dość dobrym kompromisem pomiędzy czasem obliczeń a efektem.



Rysunek 18: Wykrywanie linii

2.7 Komunikacja Aplikacja - Robot

Jednym z najważniejszych problemów przedstawionych przed twórcami projektu było zapewnienie odpowiedniej komunikacji pomiędzy aplikacją diagnostyczną, a serwerem *Urbi* obsługującym funkcjonowanie robota. W tym celu wykonano dynamiczną bibliotekę dla aplikacji oraz odpowiedni moduł *Urbi*.

2.7.1 Biblioteka dynamiczna dla aplikacji

Zdecydowano się na wykonanie dynamicznej biblioteki aby uniezależnić pracę aplikacji diagnostycznej od oprogramowania związanego z *Urbi*. Dzięki temu rozwiązaniu możliwe jest tworzenie nowych pluginów oraz modyfikacje aplikacji diagnostycznej w każdym środowisku posiadającym jedynie wpsarcie *Qt*. Biblioteka wyposażona jest w interfejs umożliwiający aplikacji wykonywanie następujących operacji:

- Nawiązanie połączenia z serwerem o danym adresie IP oraz porcie. Połączenie nawiązywane jest z wykorzystaniem protokołu TCP/IP. Umożliwiające wysyłanie komend do serwera.
- Wymuszenie nawiązanie połączenia zwrotnego do przesyłu danych z serwera do aplikacji diagnostycznej.
- Żądanie pojedynczego przesyłu danych do aplikacji.
- Żądanie synchronicznego przesyłu danych z określoną częstotliwością przesyłu.
- Zmianę częstotliwości synchronicznego przesyłu danych.
- Zatrzymanie synchronicznego przesyłu.
- Rozłączenie serwera z aplikacją diagnostyczną
- Przesłanie skryptu do sterowania robotem

- Przesłanie komendy - Uwaga komenda przesyłana jest w prostej postaci (*QString*). Aby serwer zareagował na wybraną komendę niezbędna jest implementacja tego zachowania w module podłączonym do serwera. Komendy dostępne w implementacji modułu to:
  - Dla pluginu związanego z obsługą kamery obsługiwane są komendy związane z wyborem oraz otwarciem odpowiedniego portu szeregowego przy ustawionych parametrach do sterowania kamerą oraz komendy związane z przesyłem komunikatów przez port szeregowy wykorzystując połączenie szeregowe.
  - Dla aplikacji komendy związane z rozpoczęciem działania

2.7.2 Moduł urbi

Od strony serwera urbi, na którym pracuje robot zdecydowano się stworzyć oddzielny moduł umożliwiający komunikację między robotem a aplikacją. Zadaniem tego modułu jest reagować na wszystkie polecenia opisane w wypunktowaniu powyżej. Moduł jest napisany w języku *C++* wykorzystując VS 2008 z wtyczką do kompilacji modułów urbi. Komunikacja Aplikacja-Serwer wykorzystuje możliwość przesyłania komend poprzez jeden z mechanizmów urbi jakim jest UClient. Natomiast komunikacja Serwer-Klient nawiązywana, gdy nastąpi żądanie od serwera tworzona jest poprzez *QTcpSocket*. Do synchronicznej komunikacji w urbi wykorzystywany jest mechanizm update() - czyli cykliczne wykonywanie danej funkcji, której częstotliwość zmieniana jest przez serwer w razie żądania. Dokumentacja dokładna funkcji modułów została wygenerowana przez program *Doxygen* i dołączona jako załącznik do dokumentacji.