POLITECHNIKA WARSZAWSKA
Wydział Transportu
Zakład Systemów Informatycznych i Mechatronicznych w Transporcie

GRANT REKTORSKI

nr umowy 540020200111

DLA
STUDENCKIEGO KOŁA NAUKOWEGO
ELEKTROTECHNIKI W SYSTEMACH TRANSPORTOWYCH
KNEST

BUDOWA UKŁADU TELEMETRII DO REJESTRACJI PARAMETRÓW EKSPLOATACYJNYCH POJAZDÓW ELEKTRYCZNYCH

Kierownik grantu:
dr inż. Andrzej Czerepicki

Wydział Transportu Politechniki Warszawskiej
Gmach Nowej Kreślarni, p.221,
tel. (22) 2347752, kom. (48) 664-996-179
e-mail: a.czerepicki@wt.pw.edu.pl

Warszawa 2015 r.

BUDOWA UKŁADU TELEMETRII DO REJESTRACJI PARAMETRÓW EKSPLOATACYJNYCH POJAZDÓW ELEKTRYCZNYCH

Sprawozdanie z realizacji Grantu Rektorskiego

Kierownik grantu:

- Dr inż. Andrzej Czerepicki

Opiekunowie merytoryczni:

- Dr hab. inż. Piotr Tomczuk
- Dr inż. Tomasz Dzik
- Mgr inż. Marcin Koniak

Koordynator projektu:

- Piotr Jaskowski

Studenci SKN KNEST:

- Paulina Zientek
- Ireneusz Kowalski
- Rafał Kowalski
- Łukasz Rostkowski
- Karolina Gmur
- Siergiej Mitrowski
- Sebastian Wiśniewski
- Piotr Dawidowicz
- Kamil Przybylski

Warszawa 2015
www.knest.pw.edu.pl

Spis treści

Wstęp

Celem głównym Grantu Rektorskiego realizowanego przez SKN Elektrotechniki w Systemach Transportowych KNEST był projekt i wykonanie prototypu systemu telemetrycznego do zdalnych pomiarów i rejestracji parametrów ruchu pojazdu elektrycznego.

Demonstratorem technologii jest gokart elektryczny zakupiony celem realizacji projektu w ramach Dużej Puli na projekty naukowe Rady Kół Naukowych Politechniki Warszawskiej w roku 2014 „Konstrukcja pojazdu elektrycznego gokart dostosowanego do współpracy z zewnętrznym źródłem sterowania".

Pojazdy elektryczne są obecnie najmniej liczną grupą pojazdów samochodowych w Polsce [1]. Zastosowane w nich rozwiązania elektryczne i elektroniczne pozwalają na pokonywanie niewielkich odległości, najczęściej w ruchu miejskim. Nadal istnieje ograniczenie w postaci akumulatora – zasobnika energii, którego pojemność zasadniczo definiuje zasięg pojazdu. Na ten kluczowy parametr poza pojemnością akumulatora wpływa szereg czynników związanych z zachowaniem kierowcy (stylem jazdy), ukształtowaniem terenu oraz temperaturą otoczenia.

Nadzór telemetryczny nad parametrami pojazdu znany jest np. z zastosowań w wyścigach samochodowych, czy rozwiązań transportowych w aplikacjach logistycznych [2]. Może on także znaleźć zastosowanie w zakresie monitorowania stanu technicznego pojazdu elektrycznego.

Zaproponowane rozwiązanie techniczne pozwala zarejestrować i przechować dane dotyczące kluczowych parametrów związanych z eksploatacją pojazdu elektrycznego.

Zadania projektowanego systemu:

- dokonanie pomiaru parametrów eksploatacyjnych podczas ruchu pojazdu elektrycznego w kategoriach: zasilania (napięcie na akumulatorze oraz prąd pobierany przez układ napędowy), sterowania (bieżąca prędkość, przyspieszenie/opóźnienie pojazdu), danych o temperaturze silnika oraz lokalizacyjnych - współrzędnych GPS,
- przeprowadzenie wstępnej obróbki i konsolidacji pozyskanych danych w pakiety,
- przekazanie informacji do odbiorcy drogą radiową,
- odczyt przesłanej informacji po stronie odbiorcy oraz jej archiwizacja i wizualizacja.

W pierwszym etapie prac została zaproponowania i zaprojektowana architektura rozwiązania. Następnym etapem realizacji projektu było dokonanie zakupów niezbędnych podzespołów w tym czujników i sensorów pomiarowych, zestawów do prototypowania mikrokontrolerów, radiomodemów do komunikacji bezprzewodowej oraz układu gromadzenia i prezentacji danych pomiarowych. Szczególna uwaga została zwrócona na skonfigurowanie trybu pracy czujników oraz dobraniu metod i sposobów odczytu pomiarów. W celu zbierania i przetwarzania danych pomiarowych niezbędne było napisanie

szeregu aplikacji sterujących mikrokontrolerami. W kolejnym etapie opracowano algorytm kompresji danych pomiarowych oraz protokół przesłania danych drogą radiową. Na tym etapie odbyło się konfigurowanie radiowych modułów komunikacyjnych. Następnie została opracowana aplikacja komputerowa pozwalająca na odczyt i deszyfrację danych przekazanych z pojazdu a także jej prezentację w postaci graficznej.

Testowanie systemu odbyło się na stanowiskach laboratoryjnych z wykorzystaniem aparatury badawczej, jaką dysponuje Wydział Transportu. Podjęte zostały próby testowania rozwiązania w środowisku rzeczywistym.

Tabela 1. Harmonogram prac

Miesiąc	**Zadanie**
Maj 2015	Wykonanie wstępnego projektu w postaci schematu ideowego oraz zakup komponentów do budowy urządzenia
Czerwiec 2015	Rozpoczęcie prac konstrukcyjnych nad urządzeniem. Dobór właściwych czujników oraz konfigurowanie parametrów odczytu.
Październik 2015	Programowanie aplikacji na platformach mikrokontrolerów do odczytu danych pomiarów. Wybór metody kompresji oraz protokołu przekazywania danych w celu zdalnego odczytu oraz składowania.
Listopad 2015	Zakończenie budowy prototypu urządzenia. Testy w warunkach laboratoryjnych. Opracowanie aplikacji do wizualizacji danych.
Grudzień 2015	Opracowanie sprawozdania

1. Podstawy telemetrii

Mianem telemetrii nazwano szerokie zastosowanie technik zdalnego pomiaru, przesyłania oraz archiwizacji parametrów pomiarowych na odległość [3]. W miarę postępu technicznego, stale rosła potrzeba stworzenia zdalnego pomiaru. Pierwszym współczesnym obiektem telemetrycznym było „poprowadzenie" przewodów i zainstalowanie miernika u dyspozytora, informującego o danych wielkościach. Następnie wykorzystując technikę łączności, zrezygnowano w większości z metod przewodowych na rzecz metod bezprzewodowych, zyskując większy zasięg przy jednoczesnym zachowaniu dokładności pomiaru. Układ był łatwy i prosty w budowie. Mianowicie, za pośrednictwem przebiegu nośnego, który był modulowany, transmitowano go torem komunikacyjnym i następnie po demodulacji, odczytywano wiadomość nadaną. Wykorzystując sieć telefoniczną, uzyskano zasięg na obszarze całej kuli ziemskiej. Wspomniana powyżej dziedzina techniki, niesie ze sobą wiele możliwości, zaczynając od pomiarów, kontroli czy sterowania. Jest wykorzystywana w wielu sektorach gospodarki, m.in. w transporcie, energetyce, badaniach naukowych itp. Umożliwia śledzenie pojazdów, w zakresie zarządzania procesem transportowym, jak lokalizacji pojazdu w przypadku kradzieży, sterowanie urządzeniami takimi tak sygnalizacja świetlna lub alarmowa, przesyłanie danych o wymaganych przez użytkowników parametrach, tj. temperatura silnika, przyśpieszenie czy zużycie paliwa.

1.1. Charakterystyka systemu telemetrii

Na systemy telemetryczne składają się:

- Komunikacja pomiędzy poszczególnymi punktami (sieć WAN/LAN, sieci telekomunikacji ruchomej, systemy satelitarne itp.),
- Urządzenia do pozyskiwania informacji,
- Prezentacja danych, po stronie użytkowników.

1.2. Klasyfikacja metod telemetrii

Telemetria jest dziedziną, która zajmuje się pomiarami odbywającymi się na odległość. Aspekt komunikacji, czyli z jakiego sposobu przesyłania wyników pomiarów skorzystamy zależy wyłącznie od konstruktora danego zadania. Tak więc ze względu na sposób przesyłania danych komunikację dzielimy na przewodową i bezprzewodową.

1.2.1. Komunikacja przewodowa

W pierwszym przypadku poszczególne punkty są połączone z użyciem kabli bądź współcześnie bardzo popularnych światłowodów [4]. Wydaje się, że ze względu na dobrze rozbudowaną sieć Internet, tylko tam komunikacja przewodowa znajduje zastosowanie. Posiada bowiem wiele wad. Im większe odległości pomiędzy urządzeniami, tym wyższe koszty budowy (konieczność ominięcia przeszkód

i uzyskanie pozwoleń na wszelakie prace ziemne) i utrzymania. Kolejne utrudnienie stanowiłyby różnego rodzaju zakłócenia – wymagane zastosowanie wzmacniaczy. Jednak dominująca wadą pozostaje konieczność stosowania przewodów.

1.2.2. Komunikacja bezprzewodowa

Lepszym rozwiązaniem jest komunikacja bezprzewodowa [5]. Oprócz łatwości w budowie, cechuje ją także duży zasięg. Można wyróżnić tutaj kilka rozwiązań. Podstawowym pojęciem jest sieć M2M (Machine-to-Machine) obejmująca szeroką gamę możliwości przesyłania informacji. Stanowi zbiór technologii różnego typu, które umożliwiają komunikowanie się urządzeń zaprogramowanych w celu przeprowadzenia wyznaczonego zadania. Powstały liczne kierunki współpracy pomiędzy urządzeniami, zadecydowało o tym:

- rozwój komunikacji w obszarze wymiany danych w sieci telefonii komórkowej,
- upowszechnienie komunikacji komputerowej w formie sieci IP (ang. Internet Protocol),
- spadek cen układów elektronicznych (mikrokontrolerów).

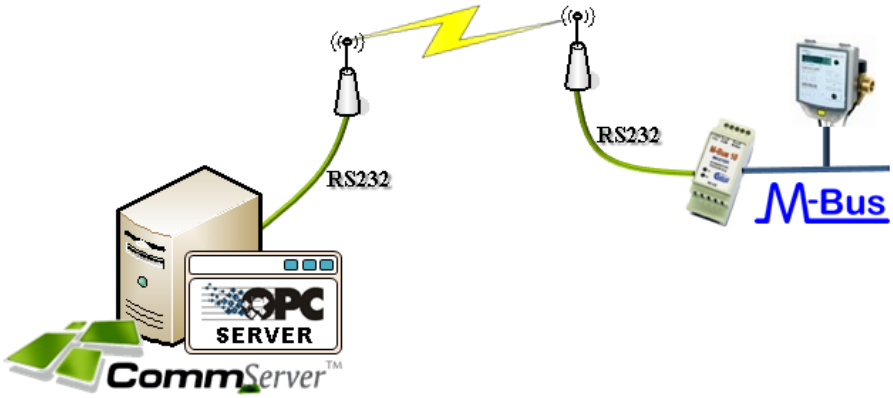
Należy pamiętać, że możliwa jest każda forma komunikacji, która pozwoli na wymianę danych pomiędzy urządzeniami. Urządzenia inteligentne tworzą sieć komunikacyjną M2M przesyłając dane dzięki standardom Bluetooth, Zigbee, Meter-Bus (M-Bus), Saia-Bus (S-Bus) czy innymi urządzeniami z interfejsem WiFi.

1.2.2.1. Radiomodemy

Jednym z popularnych rozwiązań w telemetrii jest zastosowanie radiomodemów [6]. Ich zasięg pozwala na bezprzewodową transmisję na odległość do 100 km przy szybkości od 9600 b/s do 2 Mb/s. Pracują w pasmach częstotliwości 160, 400 i 900 MHz przy pomocy implementowanych protokołów komunikacyjnych. W radiomodemach wykorzystywana jest transmisja w tzw. technologii widma rozproszonego - tzw. radiolinia. Zaletą jest niski współczynnik błędów i duża odporność na zakłócenia zewnętrzne. Urządzenia te umożliwiają przesyłanie głosu, obrazu i danych cyfrowych.

Podstawą komunikacji pomiędzy radiomodemami jest (rys. 1.1):

- nadajnik radiowy,
- odbiornik radiowy,
- modem.



Rys. 1.1. Przykład zastosowania radiomodemów (źródło: Internet).

Rynek przedstawia szeroka ofertę różnych standardów przesyłania danych. Wybór jest uzależniony od takich czynników jak: odległość pomiędzy łączonymi elementami systemu, wymaganej szybkości przesyłania danych oraz liczby sterowników i odbiorników tworzących cały system. Najczęściej stosowanymi standardami przesyłania danych są przemysłowe interfejsy szeregowe RS-232-C, RS-422-A, RS-485 oraz CAN [7].

Jeżeli wymagana jest większa szybkość transmisji przesyłanych danych wówczas zamiast radiomodemów stosuje się radiomodemy Ethernet umożliwiające pracę z szybkością 100 Mbps. Zaletą sieci wykorzystującej protokół TCP/IP [8] jest możliwość nadzorowania systemu i sterowania nim poprzez sieć Internet, czyli z dowolnego miejsca na świecie. W miarę wzrostu potrzeby istnieje możliwość zwiększenia zasięgu transmisji poprzez zastosowanie retransmiterów sygnału.

Radiomodemy umożliwiają również podłączanie do nich dodatkowych urządzeń (szereg różnych łączy). Zaletą tego rozwiązania jest niewątpliwie pełna ochrona danych – transmisja jest zabezpieczona i zaszyfrowana.

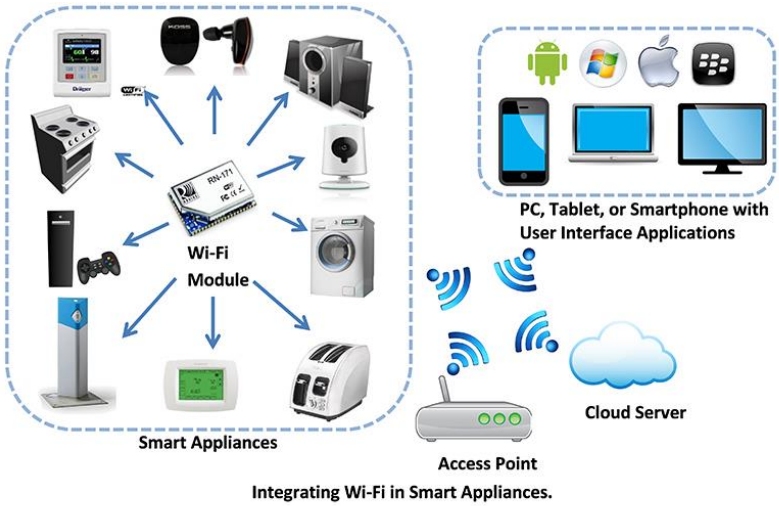
1.2.2.2. Wi-Fi

Innym sposobem umożliwiającym komunikację bezprzewodową jest standard IEEE 802.11 [9]. Są one podstawą certyfikatów Wi-Fi (rys. 1.2). Głównym zastosowaniem Wi-Fi są sieci lokalne, w których komunikacja odbywa się na drodze radiowej. Zasięg Wi-Fi sięga od kilku metrów do kilku kilometrów, a przepustowość wynosi do 300 Mb/s, zaś transmisja może odbywać się na dwóch kanałach jednocześnie. Sieć Wi-Fi działa w paśmie częstotliwości od 2400 do 2485 MHz (2,4 GHz) lub 4915 do 5825 MHz (5 GHz). Zaletą podnoszącą atrakcyjność lokalnych sieci bezprzewodowych jest duża elastyczność i łatwość jej rekonfiguracji zależnie od bieżących potrzeb użytkownika.

W sieci Wi-Fi zdefiniowano wiele różnych technologii warstwy fizycznej:

- 802.11a – 54 Mb/s częstotliwość 5 GHz.

- 802.11b – 11 Mb/s w paśmie ISM 2,4 GHz; posiada zasięg ok. 30 m w pomieszczeniu i 120 m w otwartej przestrzeni. Materiały takie jak woda, metal, czy beton obniżają znacznie jakość sygnału. Standard 802.11b podzielony jest na 14 niezależnych kanałów o szerokości 22 MHz.
- 802.11n – 100-600 Mb/s.
- 802.11g – 54 Mb/s, częstotliwość 2,4 GHz, obecnie najpopularniejszy standard WiFi, który powstał w czerwcu 2003 roku, wykorzystanie starszych urządzeń w tym standardzie powoduje zmniejszenie prędkości do 11 Mb/s.



Rys. 1.2. Przykłady zastosowania standardu Wi-Fi (źródło: Internet)

Do zalet sieci Wi-Fi należą:

- budowa sieci z dostępem do Internetu pozbawiona okablowania,
- bezprzewodowe podłączanie do sieci mobilnych urządzeń (notebooki, laptopy, telefony komórkowe),
- technologia tania i łatwo dostępna,
- duża odporność na wyładowania atmosferyczne,
- relatywnie szybka sieć w porównaniu ze standardowymi wymaganiami.

Niestety, jak w każdej technologii, są też pewne wady tego rozwiązania:

- stosunkowo mały zasięg,
- dość duża podatność na ataki hackerów,
- nie jest stosowana przy dużych odległościach,
- urządzenia, które pracują na tych samych kanałach, mogą nawzajem zakłócać swoje sygnały,
- prędkość transmisji zależy od odległości między urządzeniami komunikującymi się.

1.2.3. GSM

Intensywny rozwój systemów telefonii komórkowych i ich cyfrowej transmisji sygnałów, a także wprowadzenie przez operatorów GSM możliwości transmisji danych w standardzie GPRS oraz EDGE, UMTS, HSDPA przyczynił się do powstania możliwości sprzyjających rozbudowie systemów pomiarowych [10].

Podczas rozwoju systemu GSM zostały wyodrębnione cztery generacje:

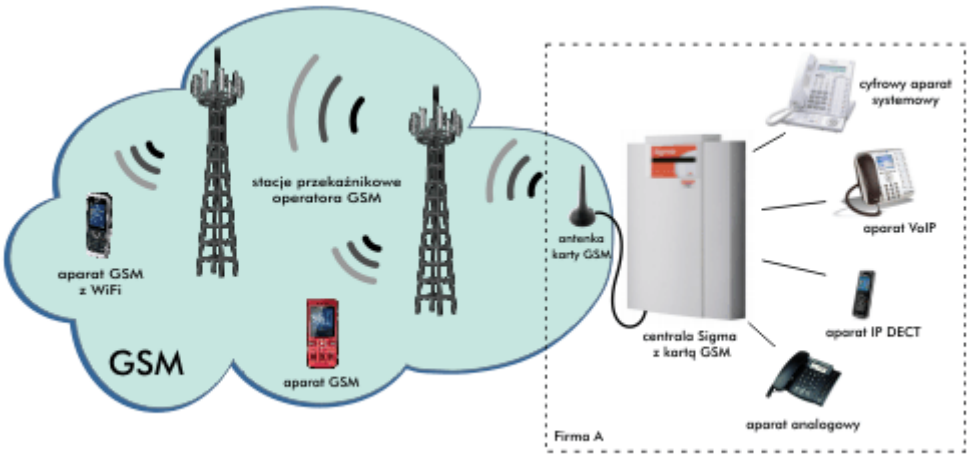
- 1G – analogowa technologia pierwszej generacji telefonii komórkowej pracująca na częstotliwościach 450 i 900 MHz . Niestety wiązała się z szeregiem wad, takich jak niski poziom bezpieczeństwa czy niewielka szybkość transmisji.

- 2G – kolejnym krokiem była technologia cyfrowa. Częstotliwości pracy były następujące: 900 i 1800 MHz, stworzono usługę, która umożliwiała przesyłanie krótkich wiadomości tekstowych (potocznie SMS), faksów oraz telekonferencję.

- 3G – następna generacja telefonii komórkowej. Pasma częstotliwości to 1950 i 2150 MHz. Zaletą tej generacji jest wprowadzenie usług pracujących dzięki transmisji video oraz transmisję pakietową.

- 4G – najnowsza technologia przesyłu danych. Pracuje w paśmie 18000 MHz, transmisja dochodzi do 150 Mb/s. Cechą szczególnie wyróżniającą tą technologię spośród poprzedniczek jest szybkość transferu pomiędzy urządzeniami. Oferuje użytkownikom mobilny Internet, zindywidualizowaną telefonię, dostęp do serwisów z multimediami.

Monitoring pojazdów samochodowych wykorzystuje najczęściej system GSM. Antena, odbiornik GPS jak i nadajnik GSM są umieszczane w pojeździe w miejscu niewidocznym. Położenie pojazdu jest śledzone przez stację monitorującą. Na rys. 1.3 przedstawiono budowę systemu GSM.



Rys. 1.3. Budowa systemu GSM (źródło: Internet)

Technologia GSM/GPRS jest doskonale dopasowana dla systemów monitoringu i telemetrii. Posiada wiele zalet, m.in.:

- jest powszechnie stosowany,
- możliwość korzystania z istniejącej struktury sieci transmisyjnej,
- posiada duży zasięg sieci,
- brak konieczności stosowania specjalnych anten.

Natomiast wadami są;

- stosunkowo duże koszty transmisji,
- niepewność w przesyłaniu danych (nieosiągalny zasięg operatora).

1.3. System nawigacji satelitarnej GPS

Obecnie jednym z najpowszechniej stosowanych systemów jest system GPS [11] – *Global Positioning System* (rys. 1.4) . Został wprowadzony przez Departament Obrony USA. Działanie tego systemu opiera się na przesyłaniu sygnału przez jedną z 24 satelitów NAVSTAR do odbiorników znajdujących się na ziemi. Ogromną zaletą tego rozwiązania jest poprawność działania w każdych warunkach atmosferycznych w sposób ciągły, w dowolnym czasie i miejscu. Zasada działania tego systemu jest bardzo prosta. Otóż do odbiornika docierają dane nawigacyjne i informacje czasowe z pewnym opóźnieniem czasowym, które zależy od odległości pomiędzy satelitą a odbiornikiem. Następnie urządzenie odbiorcze oblicza różnice pomiędzy sygnałem wysłanym a sygnałem odebranym z satelity i oblicza czasy. Owa różnica czasu jest mnożona przez prędkość fal radiowych, wynikiem jest odległość. Kolejno jest wyliczana pozycja odbiornika na Ziemi. Wystarczające są pomiary jedynie z 3 satelitów. Odbiornik samodzielnie wyznacza swoją pozycję.

Rys. 1.4. Schemat działania systemu GPS (źródło: *Internet*).

Realizacja komunikowania odbywa się przy pomocy rejestratora GPS. W skład wyposażenia stacji roboczej wchodzą czytnik danych oraz oprogramowanie.

1.4. Wybór technologii realizacji projektu

W niniejszym projekcie komunikacja odbywa się przy wykorzystaniu radiomodemu w standardzie RS232. Wybór ten ma swoje uzasadnienie:

- odporność na zakłócenia,
- nie występują dodatkowe koszty eksploatacyjne,
- duży zasięg – w porównaniu do technologii standardu 802.11,
- system pojednokrotnym skonfigurowaniu nie wymaga obsługi i po włączeniu jest gotowy do pracy.

Do odczytu danych o bieżącej pozycji pojazdu użyto czujnika GPS podłączanego do urządzenia sterującego poprzez łącze RS 232.

Odczyt danych pomiarowych bezpośrednio z czujników będzie zorganizowany przez mikrokontroler na bazie technologii Arduino ™.

Komputer pokładowy będzie skonstruowany na bazie komputera typu PC w formacie miniITX oraz umieszczony na ramie gokarta.

2. Projektowanie oraz sprzętowa implementacja układu telemetrii

2.1. Wymagania funkcjonalne systemu

Wbudowane systemy informatyczne, do jakich należy odnieść projektowany system, coraz częściej są stosowane na całym świecie, także w pojazdach samochodowych. Jednym z czynników mających na to wpływ, jest dostępność platform prototypowania mikrokontrolerów. Pozwalają one na budowę rozwiązań sprzętowych w połączeniem z zaawansowanymi technikami programowania, które oferują współczesne języki programowania wysokiego poziomu.

Układ telemetrii powinien:

- Realizować pomiar parametrów jazdy gokarta elektrycznego w czasie rzeczywistym.
- Dokonywać zapisu odczytanych danych do plików dziennika na komputerze pokładowym gokarta.
- Przekazywać odczytane dane drogą radiową do komputera zdalnego.
- Dokonywać wizualizacji odczytanych danych na komputerze zdalnym w postaci graficznej.

Do rejestrowanych charakterystyk jazdy należą:

- Prędkość oraz przyspieszenie pojazdu,
- Współrzędne geograficzne pojazdu,
- Temperatura wybranych układów (np. bateria, silnik),
- Napięcie na akumulatorze,
- Prąd akumulatora.

Zakres wielkości mierzonych charakterystyk przedstawia tabela 2.

Tabela 2. Rejestrowane parametry w czasie ruchu pojazdu

Parametr	**Jednostka miary**	**Wartość minimalna**	**Wartość maksymalna**
Współrzędne GPS	[geograficzne °' "]	nie dotyczy	nie dotyczy
Prędkość bieżąca	$[\frac{km}{h}]$	0	90
Przyspieszenie	$[\frac{m}{s^2}]$	0	20

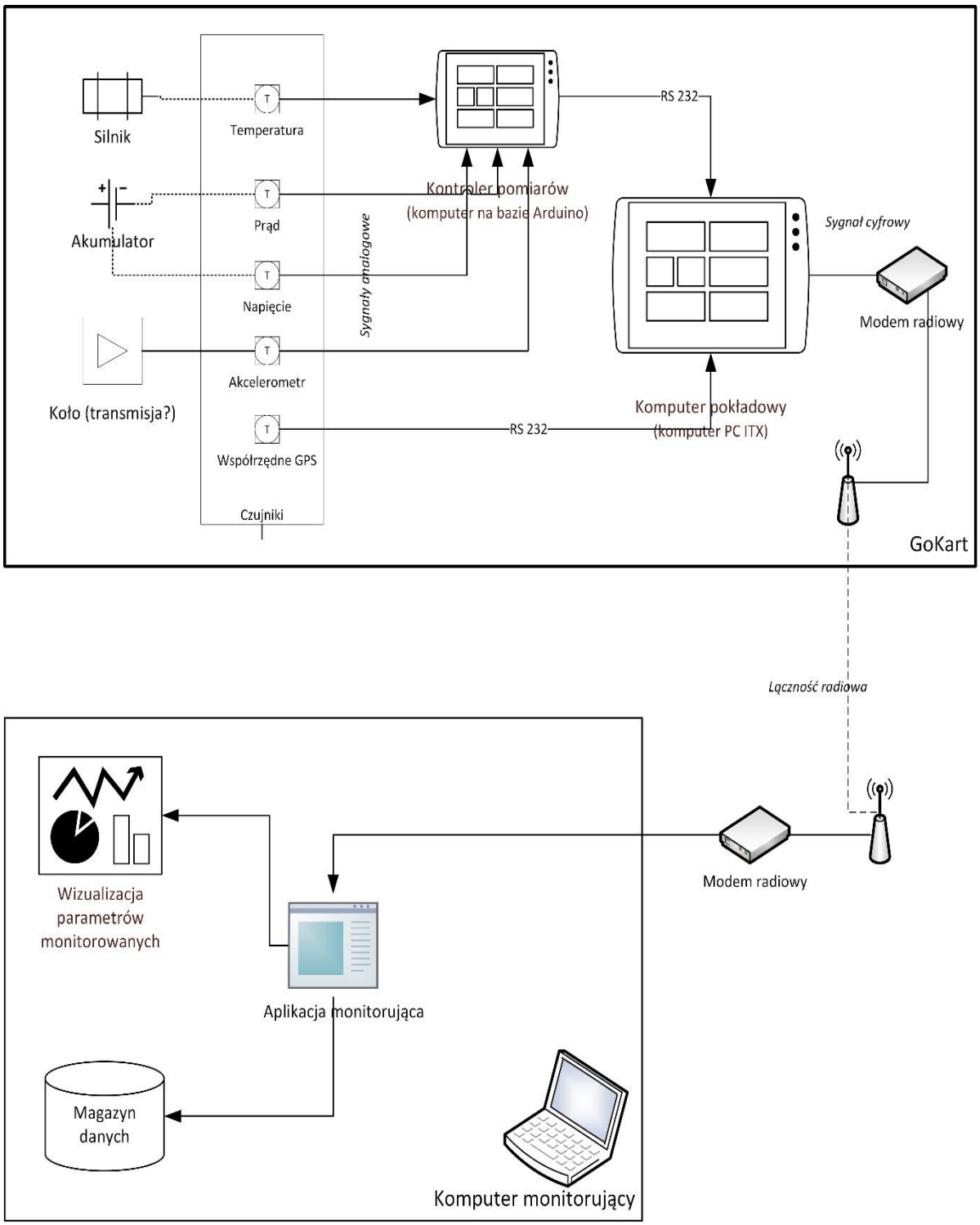
Parametr	Jednostka miary	Wartość minimalna	Wartość maksymalna
Prąd	[A]	0	150
Napięcie na akumulatorze	[V]	40	60
Temperatura silnika	st. C	20	160
Temperatura przekształtnika	st. C	20	160

2.2. Koncepcja funkcjonowania oraz architektura systemu telemetrycznego

System składa się z dwóch komputerów oraz mikrokomputera pomiarowego (rys. 2.1) . Zadaniem komputera pokładowego jest odczyt danych z czujników oraz sensorów, ich transformacja, walidacja oraz integracja w pakiety w celu przesłania do komputera zdalnego. Transmisja danych odbywa się drogą radiową z wykorzystaniem modemów radiowych.

Komputer zdalny służy do wizualizacji przesłanych danych w postaci graficznej. Pomiary charakterystyk jazdy odbywają się za pomocą czujników. Ze względu na różnorodność mierzonych wielkości fizycznych, ich odczyt może być dokonany bezpośrednio przez komputer pokładowy, lub też za pomocą kontrolera pomiarów – w przypadku, gdy wymagana jest wstępna obróbka danych.

Komunikacja z czujnikami danych odbywa się poprzez interfejs RS232. Komputer pokładowy jest komputerem klasy PC, zintegrowany z płytką kontrolera pomiarów na bazie mikrokontrolera Arduino ™.



Rys. 2.1. Ogólna architektura systemu telemetrii

2.3. Wybór platformy kontrolera pomiarów

Kontroler pomiarów wykonano z wykorzystaniem platformy Arduino ™ [12]. Platforma ta charakteryzuje się dość zaawansowanymi możliwościami z jednej strony a niskim zapotrzebowaniem na energię oraz kompaktową formą z drugiej. Istotną rolę odgrywają również powszechna dostępność

komponentów platformy oraz ich relatywnie niska cena w stosunku do rozwiązań profesjonalnych. Kolejnym atutem takiego rozwiązania jest możliwość łatwej integracji w ramach platformy urządzeń końcowych elektrycznych (czujniki, przełączniki etc.) z mikrokontrolerem pomiarowym.

Rozwiązanie oparte jest o platformę programistyczną Arduino Mega 2560 z 8 KB pamięci operacyjnej, 16 wejściami analogowymi i pamięcią Flash 256 KB wraz z komputerem pokładowym. Komputerem pokładowym zainstalowanym na gokarcie jest Mitac PD12TI Mini-ITX, wybrany ze względu na potrzebną ilość portów RS-232 oraz deklarowaną przez producenta energooszczędność. Dla kompatybilności działania czujników i sterownika z komputerem pokładowym pod względem napięciowym, zastosowano układ MAX232 umożliwiający konwersję napięć portów RS-232 (± 15V) do poziomu logiki tranzystorowo-tranzystorowej TTL z której składa się Arduino (±5V). W tym wypadku Mega może pracować przy napięciu 7-12V.

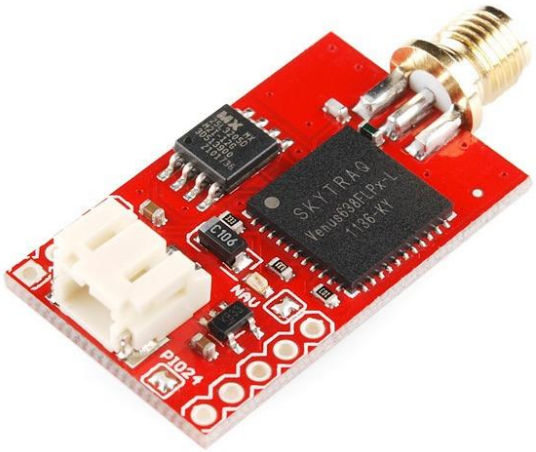
Pod uwagę brano również platformę Raspberry Pi 2 [13], została ona jednak odrzucona ze względu na małą ilość portów COM oraz problematyczną w rozwiązaniu obsługę równoległego przekazu danych. Ze względu na uniwersalność oraz szeroką gamę zastosowań w projektach edukacyjnych brano pod uwagę platformę myRIO [14]. Niestety rozwiązanie takie wymaga każdorazowego użycia myRIO wraz z komputerem PC, co zostało uznane za wadę.

Dane w postaci sygnałów analogowych są mierzone dzięki czujnikom inercyjnym, takimi jak Razor IMU pozwalający na pomiar przyspieszenia, pola magnetycznego oraz prędkości kątowej, czujnikom prądowym Hass 200-S oraz przez odbiornik GPS.W projekcie wykorzystano modemy i anteny, aby drogą radiową przesyłać informacje do komputera, który monitoruje, umożliwia wizualizacje oraz magazynuje zebrane dane.

2.4. Czujniki sygnałów

2.4.1. Odbiornik GPS

GPS Venus Logger (rys.2.2) jest odbiornikiem GPS o wysokiej szybkości odświeżania wynoszącej 20Hz. Posiada złącze SMA do podłączenia zewnętrznej anteny polepszającej odbiór sygnału. W odbiorniku tym wbudowany jest wzmacniacz LNA oraz może być zasilany napięciem od 3,5V do 12V. Jego zaletami są niewielkie wymiary (32x20mm) oraz stosunkowo duża dokładność wynosząca 2,5m. Parametry czujnika zostały zamieszczone w tabeli 3.



Rys. 2.2. Czujnik GPS

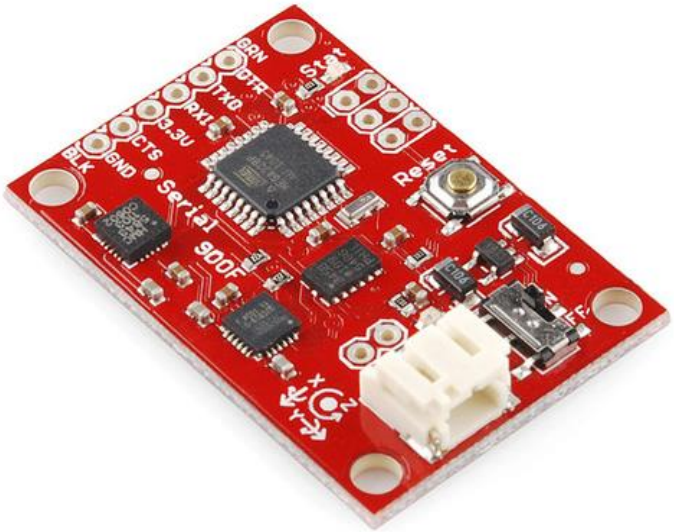
Tabela 3 Najważniejsze charakterystyki urządzeń pomiarowych

Nazwa urządzenia	**Jednostka**	**Zakres**	**Wzór**	**Mierzona wielkość**
Żyroskop	$[^{\circ}/_{s}]$	$\pm$ 2000 $^{\circ}/_{s}$	$\frac{X}{58{,}18}$	Prędkość obrotowa
Magnetometr	[T]	$\pm$ 4.7 Ga	$\frac{10^4 * X}{390}$	Indukcja magnetyczna
Akcelerometr	[g- wielokrotność przyspieszenia]	$\pm$ 2 g	$\frac{X}{256}$	Przyspieszenie
Odbiornik GPS	[geograficzne ° ' "]	-	Ramka GPGGA	Współrzędne
Termistor	$[C^{\circ}]$	$-55{\sim}125\ C^{\circ}$	-	Temperatura
Czujnik prędkości	$[^{km}/_{h}]$	0-90 $^{km}/_{h}$	$\frac{0{,}87}{X * 10^4} * 36$	Prędkość
Czujnik prądu	[A]	$\pm$ 600 A	-	Natężenie

X - mierzona wartość (liczba całkowita)

2.4.2. Akcelerometr

Czujnik Razor IMU [15] (rys.2.3) jest połączeniem trzech różnych czujników 3-osiowych: żyroskopu cyfrowego, akcelerometru i magnetometru. Moduł ten jest kompatybilny z Arduino. Dane są przetwarzane przez mikrokontroler Arduino i wysyłane poprzez interfejs szeregowy UART (RX,TX). Moduł pozwala na pomiar przyspieszenia, pola magnetycznego oraz prędkości kątowej.



Rys. 2.3. Czujnik Razor IMU

2.4.3. Czujnik prądu

Czujnik prądu Hass 200-S (rys.2.4) służy do pomiaru prądów stałych, impulsywnych i zmiennych. Do pomiarów wykorzystuje efekt Halla [16]. Charakteryzuje się dużą dokładnością +/- 1%.

Rys. 2.4. Czujnik prądu

2.4.4. Czujnik prędkości

Hallotronowy (rys.2.5) zbliżeniowy czujnik prędkości w połączeniu z magnesami może pełnić rolę uniwersalnego czujnika prędkości. Musi być zamontowany w odległości do 10 mm ze względu na strefę działania.

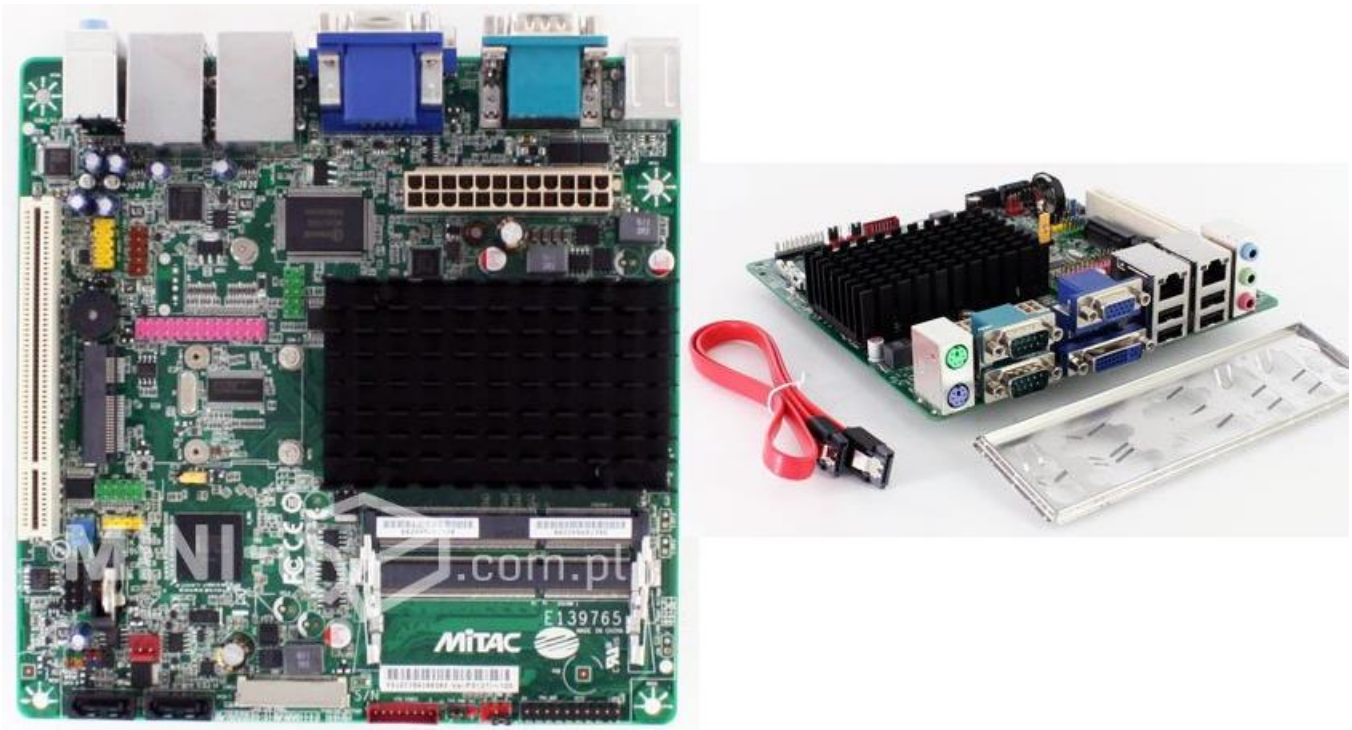
Rys. 2.5. Czujnik prędkości

2.5. Komputer pokładowy

W projekcie występuje duża różnorodność podzespołów. Poniżej przedstawiono charakterystyki najistotniejszych elementów.

2.5.1. Płyta główna komputera pokładowego

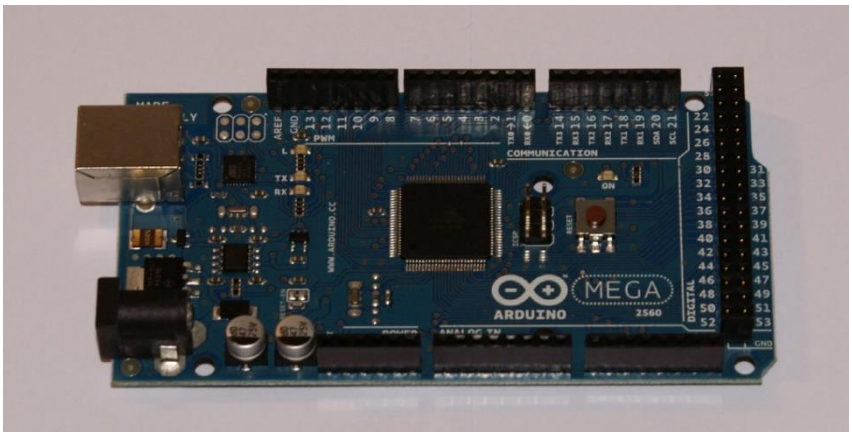
Płyta główna komputera pokładowego Mitac PD12TI formatu Mini ITX [17] jest wyprodukowana przez podwykonawcę firmy Intel (rys.2.6). W płycie jest wykorzystany energooszczędny zintegrowany procesor Intel Atom D2500, dzięki czemu wystarcza zastosowanie chłodzenia pasywnego. Układ sieciowy to Dual Intel 82574L 10/100/1000 Mb/s, za grafikę odpowiada zintegrowana karta graficzna Intel GMA 3600. Jedną z najistotniejszych zalet tej płyty jest posiadanie 4 portów COM (RS-232). Dane przetwarzane będą w pamięci RAM Kingston HyperX SO-DIMM o pojemności 4GB.



Rys. 2.6. Komputer pokładowy Mitac PD12TI

2.5.2. Kontroler pomiarów

Kontroler pomiarów został wykonany na bazie mikrokontrolera Arduino Mega (rys.2.7). Płytka jest wyposażona w mikrokontroler ATmega2560 [18], zawierający 54 cyfrowe wejścia/wyjścia z czego 15 można wykorzystać jako wyjścia PWM. Wydajność prądowa jednego wyprowadzenia wynosi 40 mA.



Rys. 2.7. Kontroler pomiarów

2.5.3. Zasilacz

Zasilacz PICO PSU DC/DC o napięciu 12V oraz mocy 120W ma dużą sprawność osiągającą nawet 95% i jest idealny do zasilania płyt Mini ITX. Dzięki swoim miniaturowym wymiarom umożliwia montaż w każdej obudowie. Istotną zaletą jest to, że generuje niewielką ilość ciepła dlatego nie ma konieczności montowania przy nim chłodzenia. Jego wymiary wynoszą 31x44x21 mm.

2.5.4. Przetwornica napięcia

Przetwornica napięcia 48/12V PO-V4 jest przeznaczona do zasilania odbiorników o poborze prądu do 4A z napięcia 48V prądu stałego. Do zalet tego urządzenia należą: duża sprawność (do 90%), małe gabaryty, niski poziom tętnień napięcia wyjściowego. Dzięki zastosowanemu radiatorowi w postaci obudowy zbędny jest wymuszony obieg powietrza dla temperatur otoczenia do 45°C. Układ posiada termiczne zabezpieczenie przeciążeniowe oraz zabezpieczenie zwarciowe.

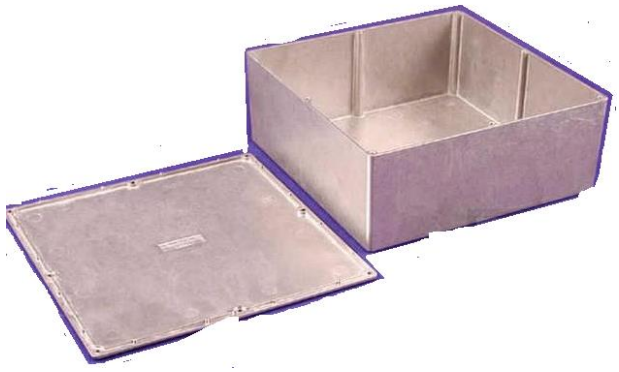
Dane techniczne przetwornicy:

- Napięcie wejściowe 36-56V,
- Napięcie wyjściowe 12V,
- Prąd wyjściowy 4A,
- Pobór prądu w stanie spoczynku 20mA,
- Zabezpieczenie termiczne - 70 stopni C,
- Zabezpieczenie zwarciowe – tak,

- Dopuszczalne tętnienia napięcia wyjściowego dla mocy nominalnej <50mV,
- Wymiary: 100 x 40 x 20 mm.

2.5.5. Obudowa

Obudowa, która została zastosowana w projekcie, ma wymiary 250x250x100mm i jest wykonana z aluminium. Zabezpiecza ona najważniejsze podzespoły przed uszkodzeniami mechanicznymi oraz czynnikami atmosferycznymi. Umożliwia też montaż wszystkich komponentów w jednym miejscu. Obudowa została zaprojektowana i wykonana z przygotówki aluminiowej przedstawionej na rys.2.8.



Rys. 2.8. Przygotówka obudowy komputera pokładowego

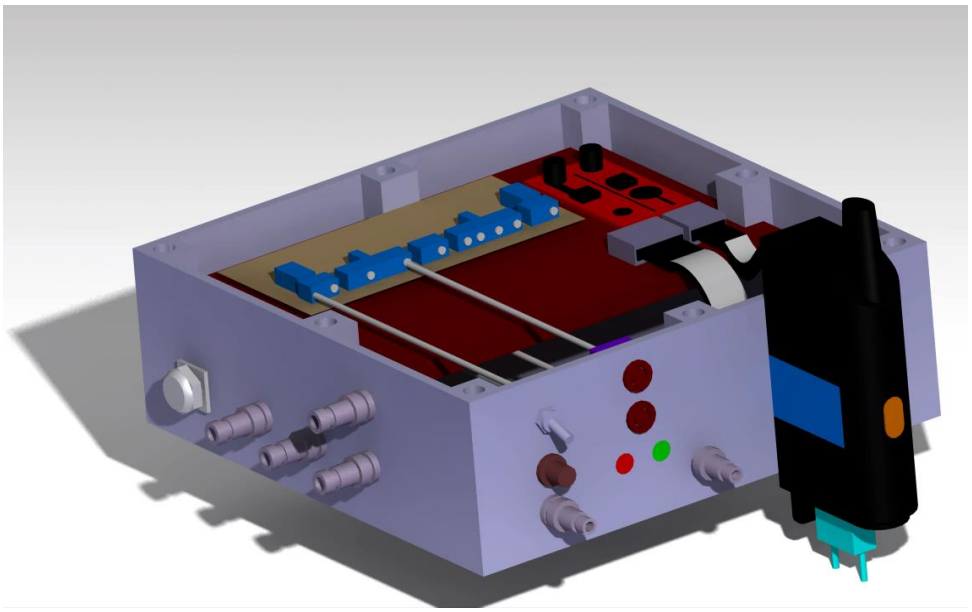
2.6. Radiomodemy

Radiomodem ARF868 [19] umożliwia dwukierunkową, bezprzewodową transmisję danych half duplex na odległość do 7km (rys.2.9). Podłączany za pomocą portu RS232 osiąga dużą prędkość radiową od 2,4 do 115,2 kbps. Działa na częstotliwości 863—870 MHz.

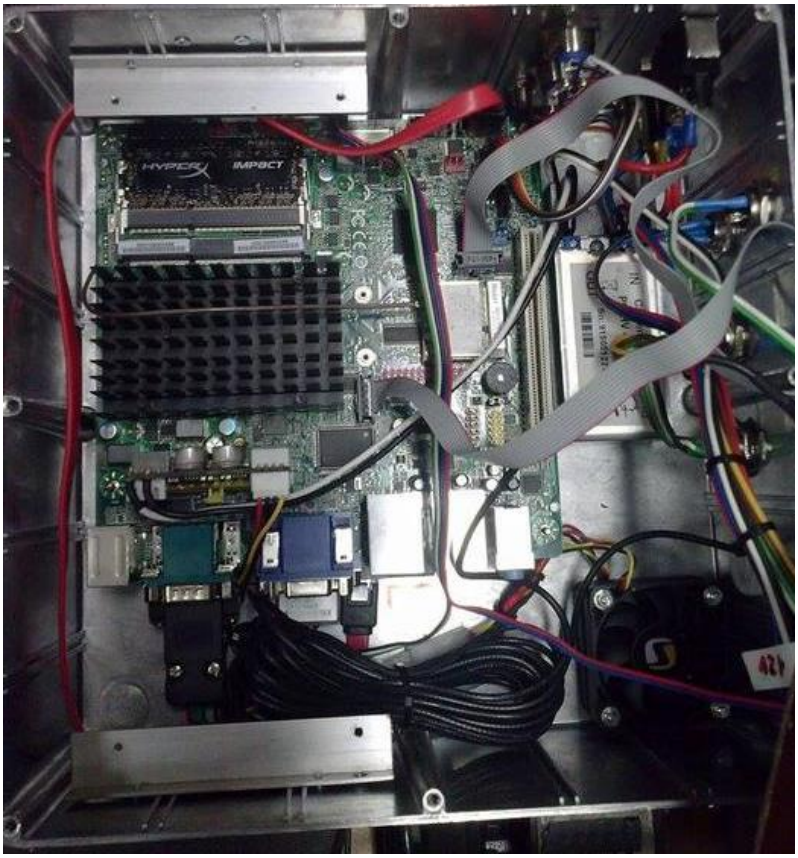
Rys. 2.9. Radiomodem ARF868

2.7. Budowa układu

Komputer pokładowy składa się z minikomputera ITX, kontrolera pomiarów na bazie mikrokontrolera Arduino, oraz modemu radiowego. Całość mieści się w aluminiowej obudowie, która posiada złącza do podłączenia zewnętrznych źródeł sygnałów (rys. 2.10, 2.11).

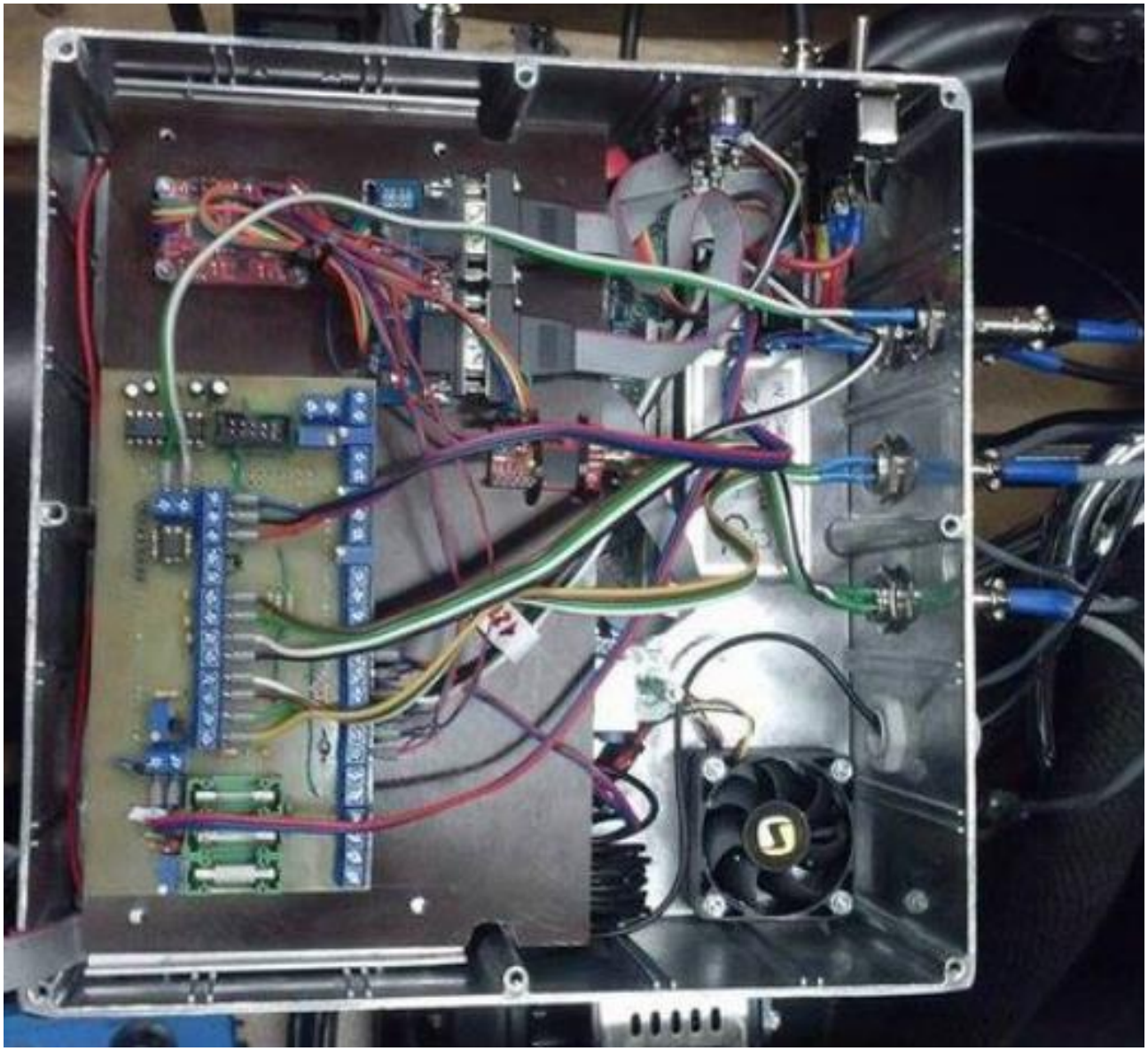


Rys. 2.10. Makieta komputera pokładowego



Rys. 2.11. Montaż płyty mini ITX w komputerze pokładowym gokarta

Powyżej płyty miniITX została umieszczona płyta kontrolera pomiarów wykonana wg projektu indywidualnego (rys. 2.12).



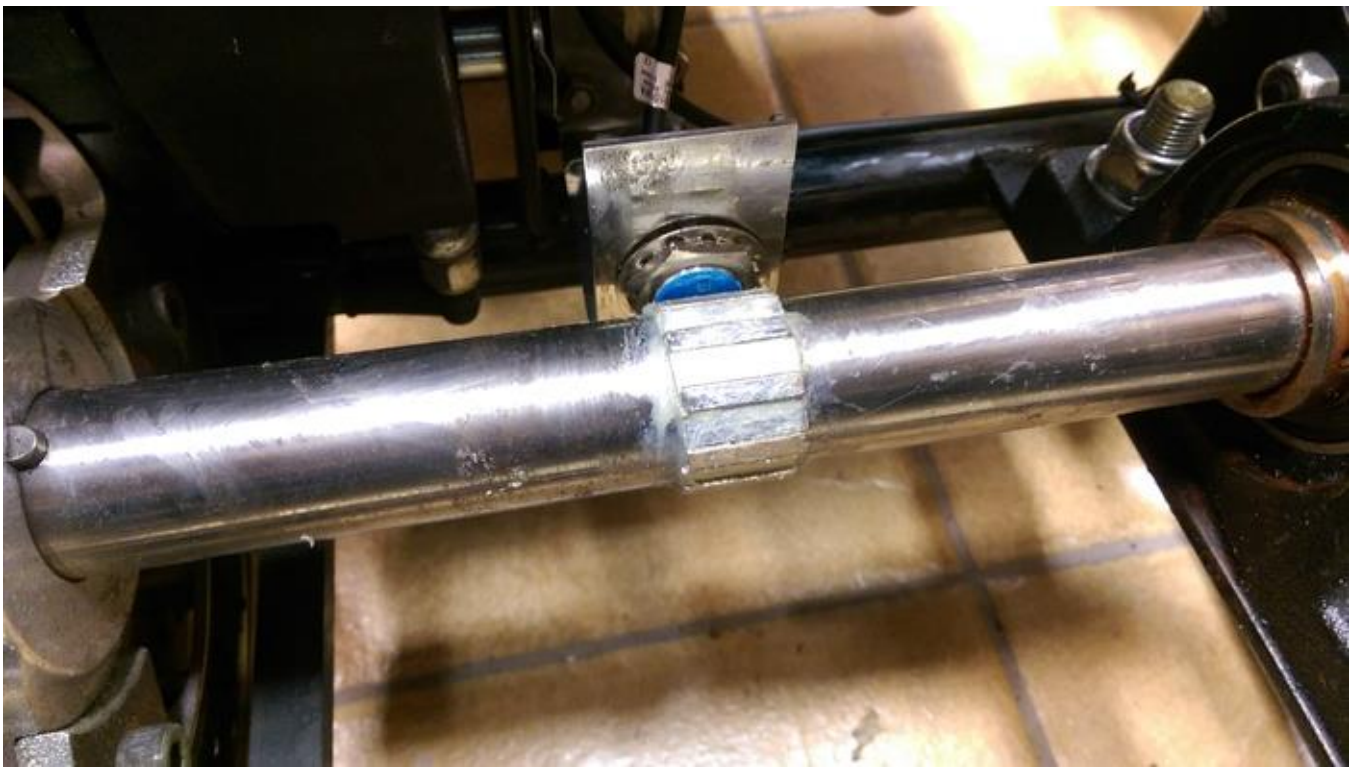
Rys. 2.12. Płyta kontrolera pomiarów

Obudowa komputera pokładowego została umieszczona na tylnej ramie gokarta, na samodzielnie wykonanej podporze (rys. 2.13) . Wewnątrz obudowy znajdują się:

- płyta kontrolera pomiarów,
- zasilacz 48/12V,
- czujnik GPS ,
- czujniki inercyjne, zasilane z płyty Arduino napięciem 5V,
- układ chłodzenia,
- złącza.

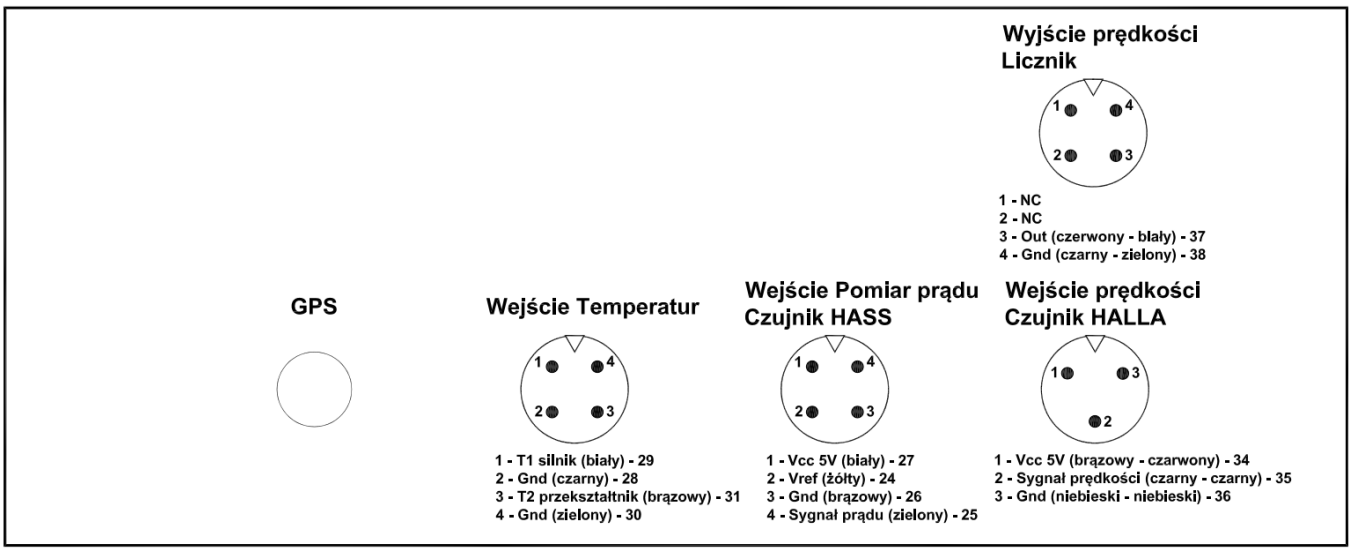
Rys. 2.13. Montaż obudowy na tylnej ramie gokarta

Czujnik Halla wykorzystywany w układzie pomiaru prędkości, jest umieszczony na tylnej osi pojazdu (rys.2.14)



Rys. 2.14. Czujnik Halla

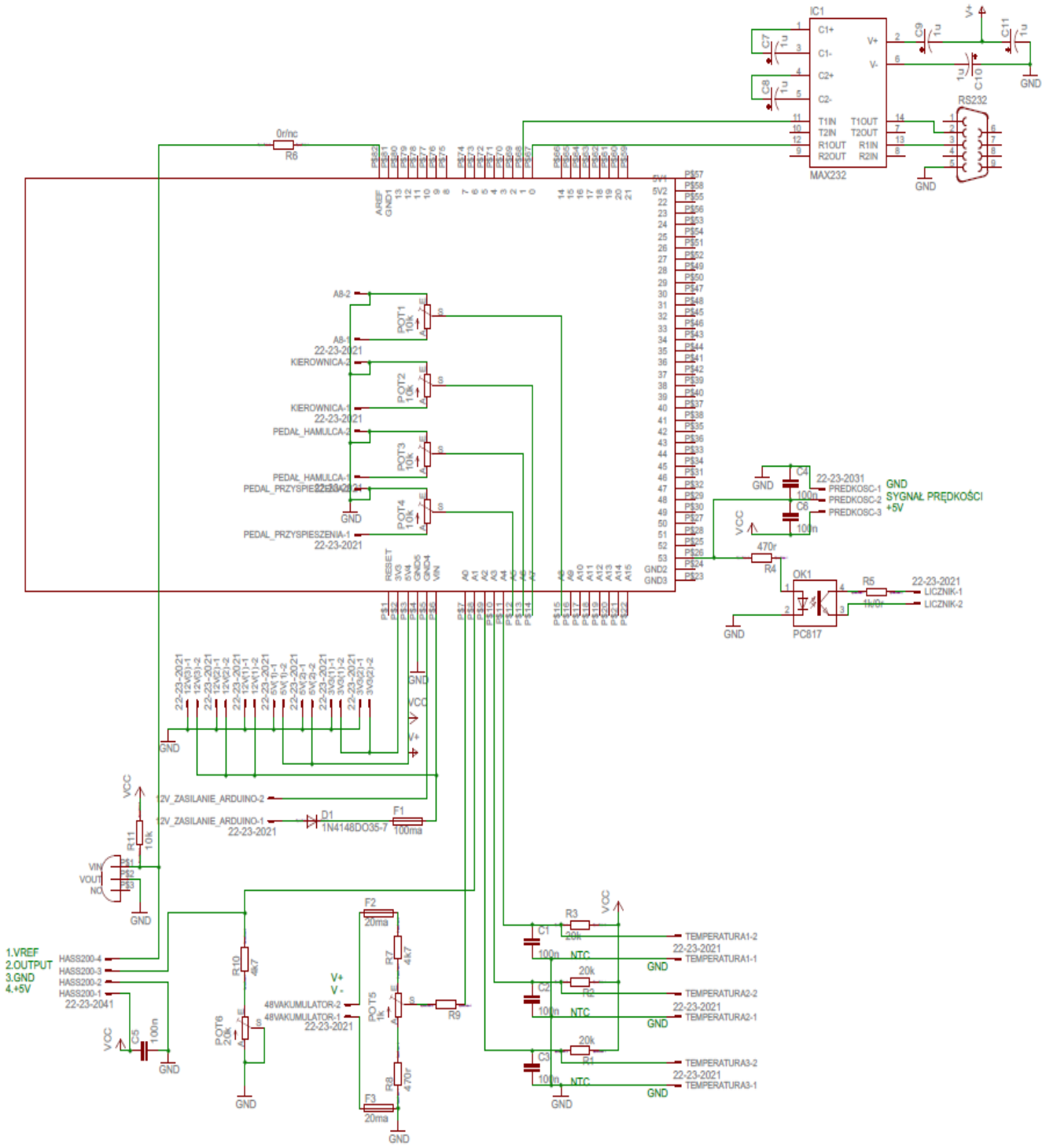
Rys. 2.15. ukazuje schemat umiejscowienia złączy do podłączenia urządzeń zewnętrznych (czujniki temperatur, czujnik prędkości, czujnik GPS) na tylnej stronie obudowy komputera pokładowego.



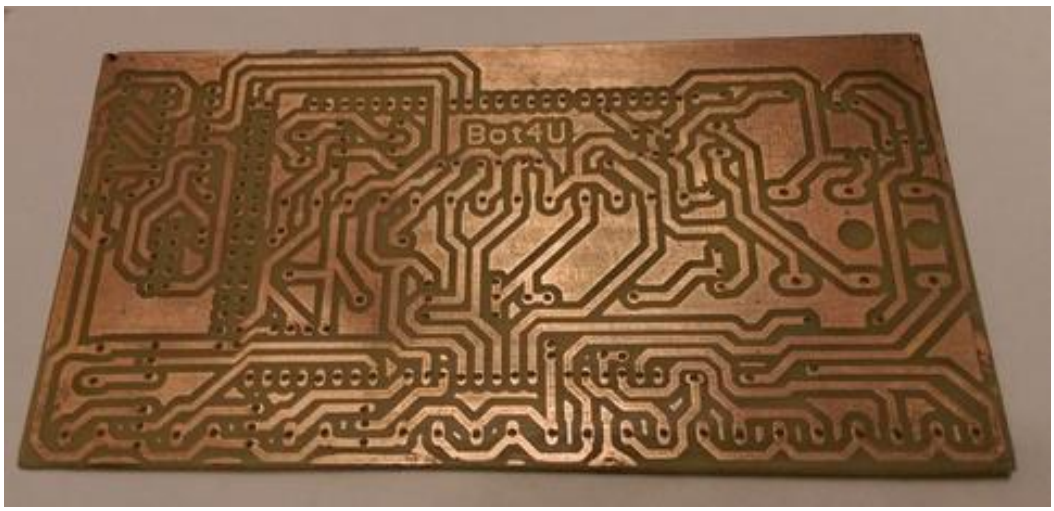
Rys. 2.15. Opis złączy zewnętrznych na obudowie

2.8. Projektowanie płytki kontrolera pomiarów

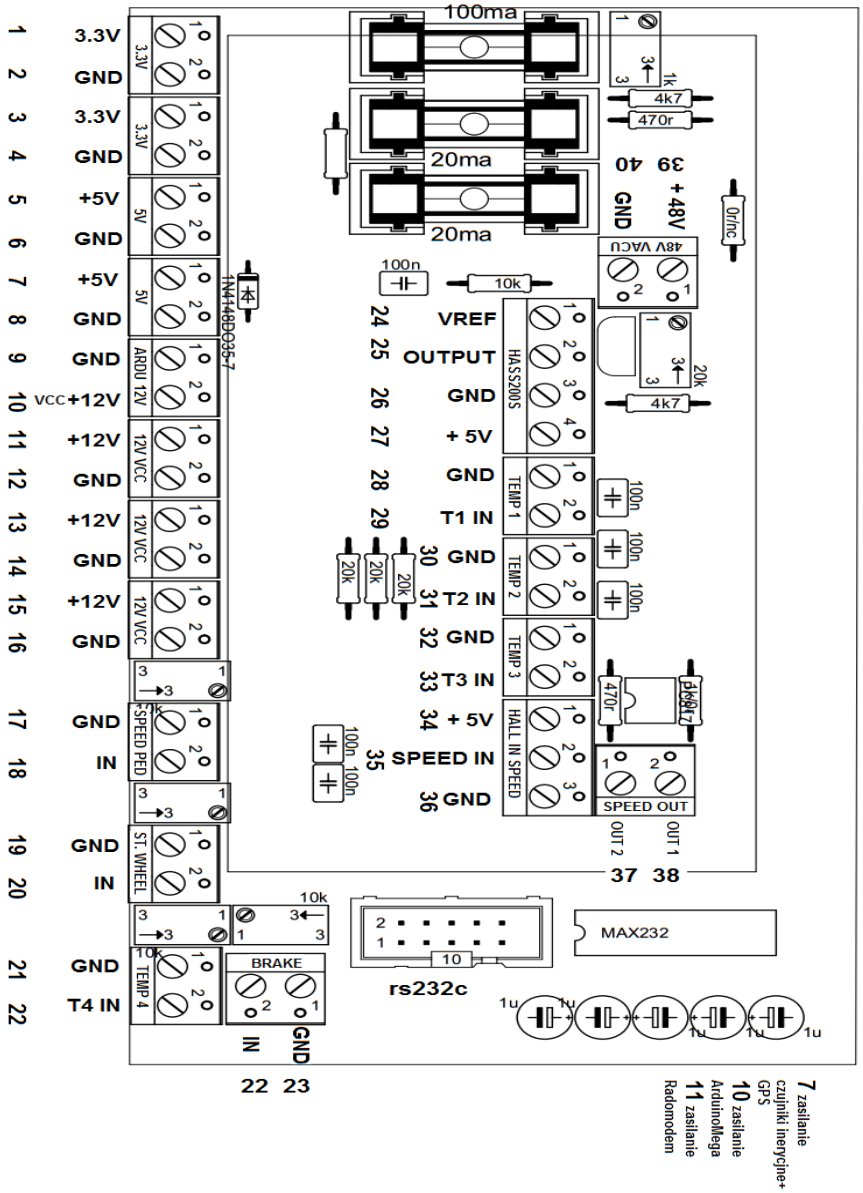
Płytka kontrolera (rys.2.16, 2.17, 2.18) pomiarów przesyła dane ze wszystkich czujników inercyjnych oraz GPS do komputera pokładowego, następnie są one przetwarzane przy pomocy napisanego na potrzeby projektu oprogramowania.



Rys. 2.16. Schemat ideowy kontrolera pomiarów



Rys. 2.17. Wytworzona płytka kontrolera pomiarów

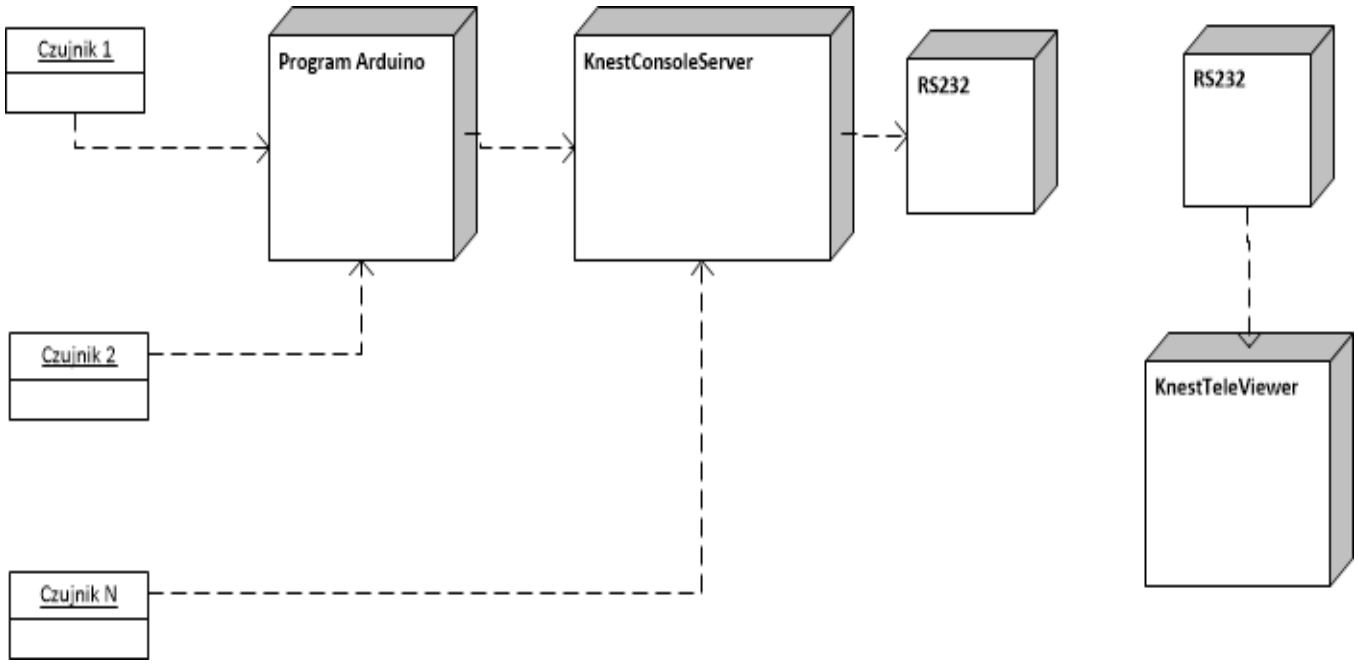


Rys. 2.18. Złącza kontrolera pomiarów

3. Projektowanie oprogramowania

Zgodnie z koncepcją opracowaną na etapie projektowania systemu, oprogramowanie składa się z następujących samodzielnych komponentów (rys. 3.1.):

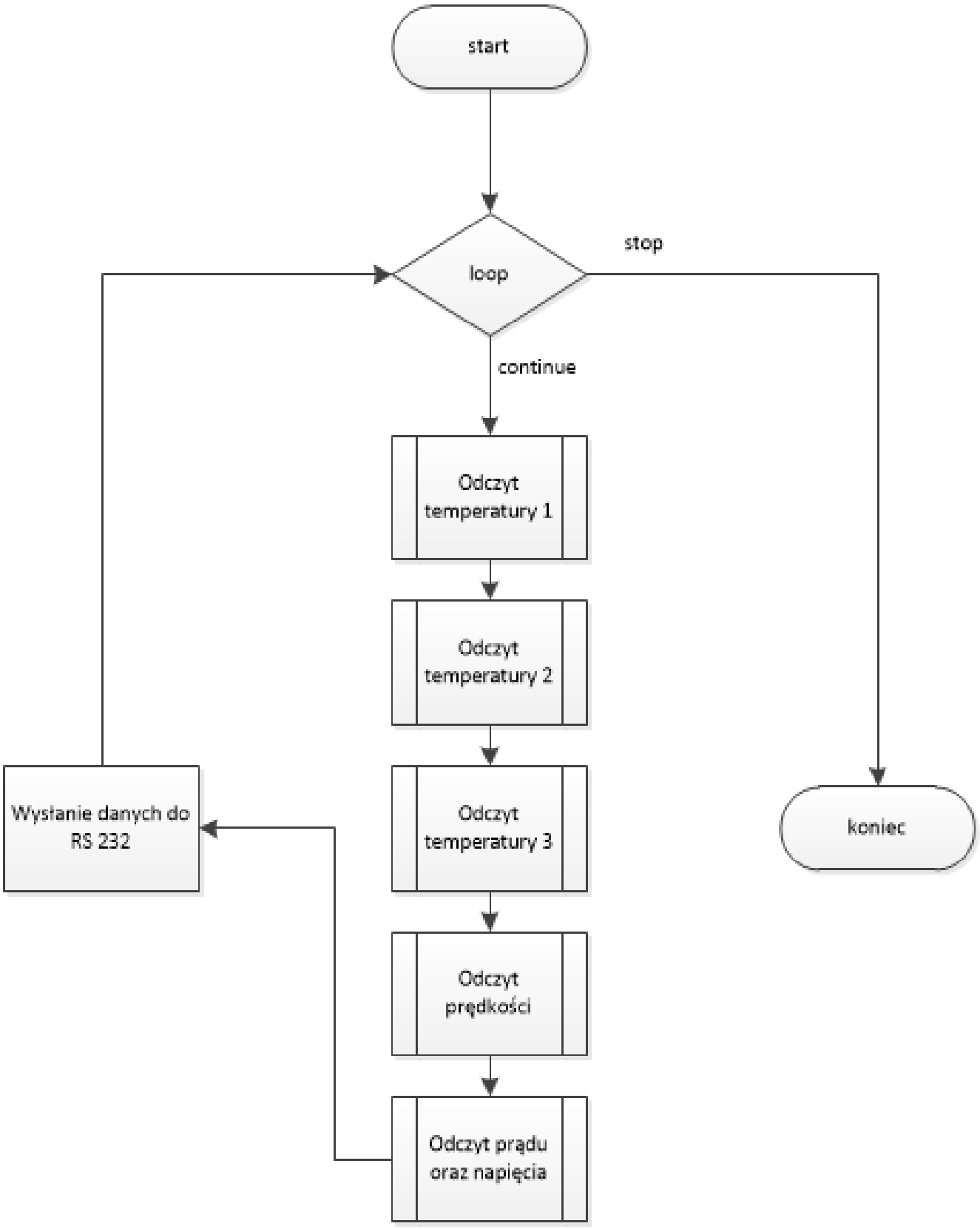
- program sterujący mikrokontrolerem Arduino – niskopoziomowy odczyt danych z czujników umieszczonych na gokarcie, oraz ich wstępna obróbka,
- program *KnestConsoleServer* – odczyt danych z kontrolera Arduino, ich konsolidacja oraz przesłanie do modemu radiowego,
- program *KnestTeleViewer* – wizualizacja danych odczytanych z modemu, oraz ich gromadzenie.



Rys. 3.1. Diagram komponentów programowych systemu

3.1. Oprogramowanie układu pomiarowego Arduino

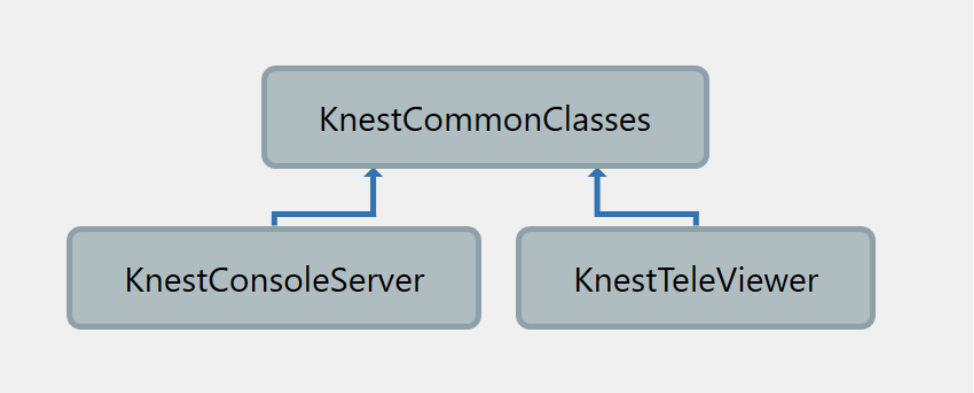
Zadaniem układu kontrolera pomiarów jest odczyt danych z czujników prędkości, temperatury, prądu itp., ich wstępna obróbka oraz przesłanie do portu RS 232 w celu przekazania danych do komputera pokładowego. Program został napisany w języku C z wykorzystaniem zintegrowanego środowiska programowania Arduino IDE. Poszczególne odczyty danych z czujników zostały opracowane jako osobne procedury (rys. 3.2) . Pętla *loop* jest wykonywana od momentu włączenia mikrokontrolera do jego wyłączenia. Kod źródłowy programu przedstawia Załącznik 1.



Rys. 3.2. Algorytm działania programu odczytu danych z czujników

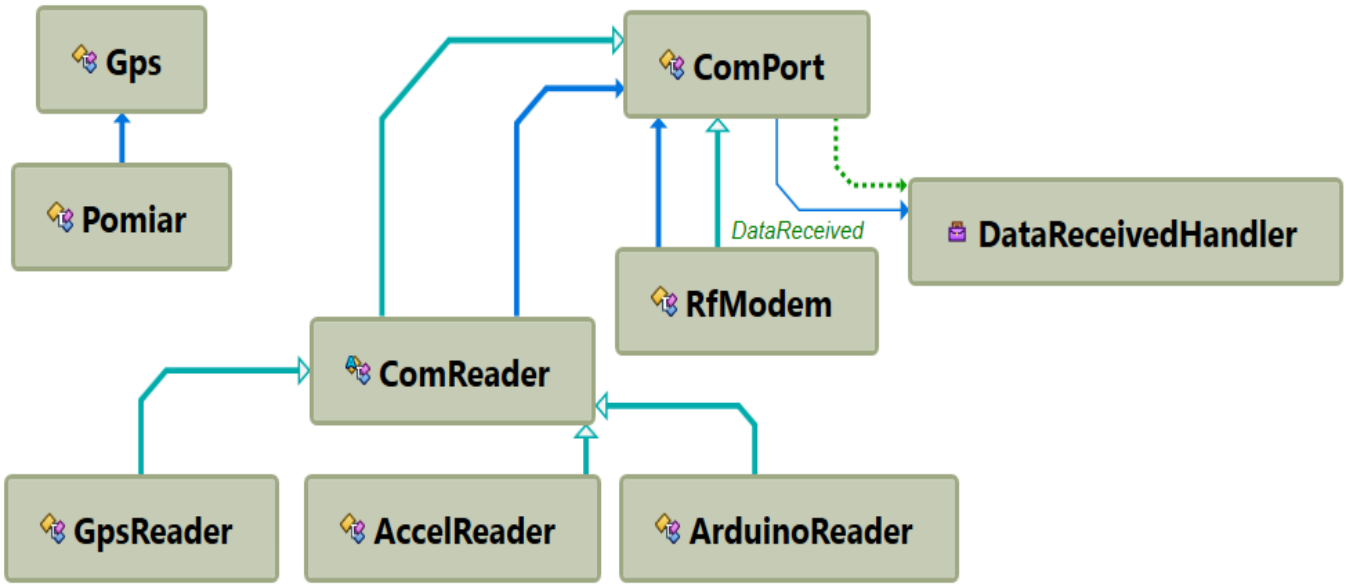
3.2. Biblioteka KnestCommonClasses

Biblioteka *KnestCommonClasses* zawiera klasy wspólne dla wszystkich programów w projekcie. Klasy te są wykorzystywane przez aplikacje *KnestConsoleServer* oraz *KnestTeleViewer* (rys. 3.3).



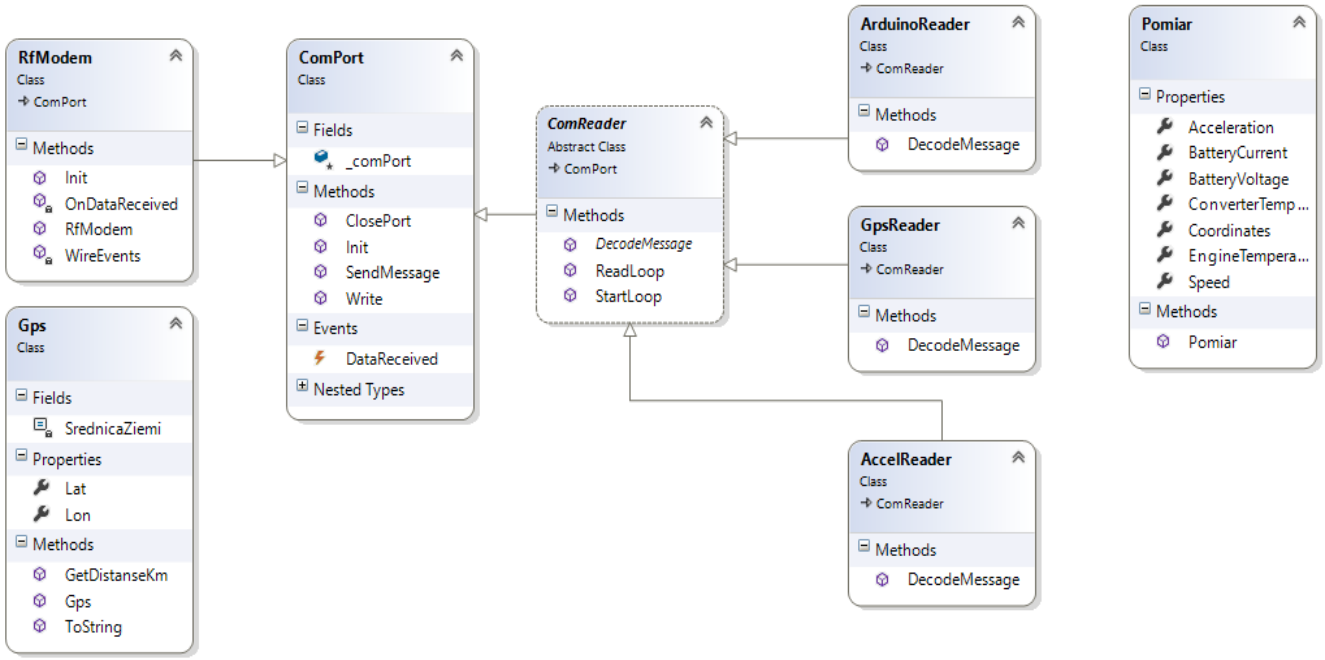
Rys. 3.3. Zależności pomiędzy komponentami w projekcie *KnestTelemetry*

Listę klas oraz ich wzajemne zależności przedstawia rys. 3.4.



Rys. 3.4. Współdziałanie klas w bibliotece *KnestCommonClasses*

Klasa *ComPort* realizuje funkcjonalność odczytu oraz zapisu danych do portów RS232, a także umożliwia ich konfigurację. Klasa *RfModem* dziedziczy z *ComPort*, rozszerzając jej funkcjonalność o asynchroniczny odczyt i zapis danych w formacie, przekazywanym do modemu radiowego. Wymiana danych odbywa się po wystąpieniu zdarzenia *DataReceived*. Klasa *ComReader* implementuje odczyt danych z portu szeregowego jako osobny wątek (ang. *Thread*). Dziedziczące z niej klasy *GpsReader*, *AccelReader* oraz *ArduinoReader* realizują odpowiednio funkcjonalność odczytu danych specyficzną dla poszczególnych rodzajów czujników. Klasa Pomiar zawiera pakiet danych przekazywany do modemu. Pola oraz metody klas stanowiących bibliotekę *KnestCommonClasses* ilustruje rys. 3.5.

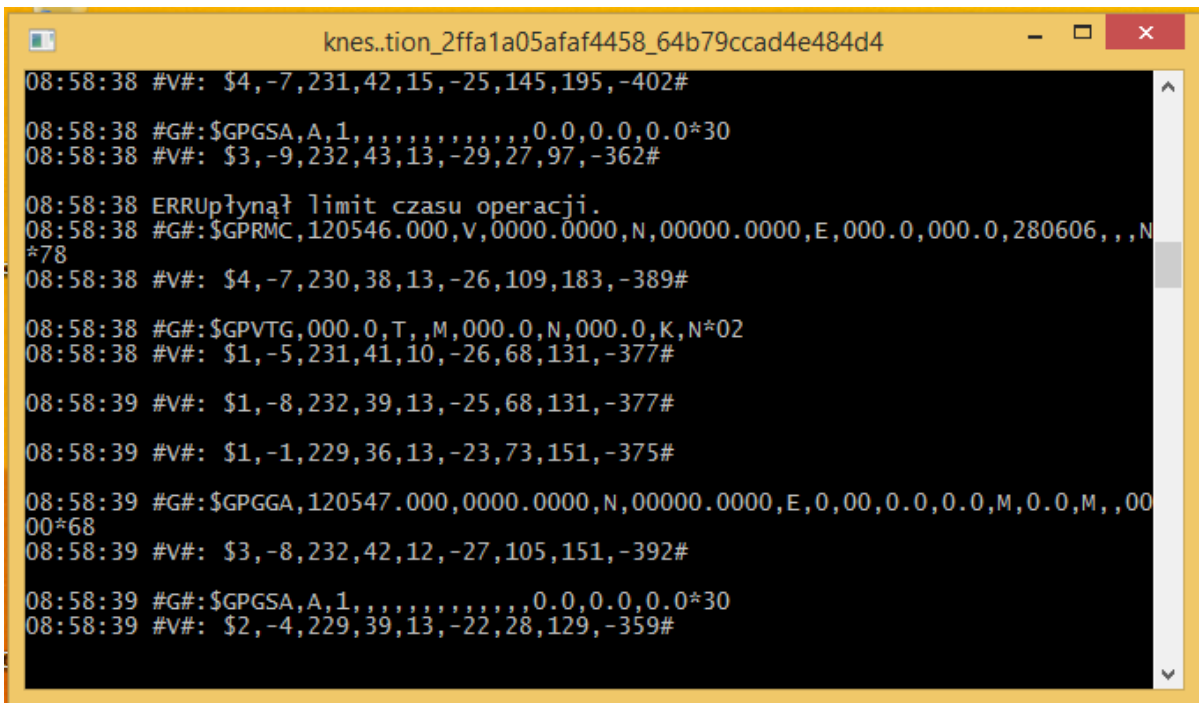


Rys. 3.5. Diagram klas biblioteki *KnestCommonClasses*

Kod źródłowy klas w bibliotece stanowi Załącznik 2.

3.3. Program KnestConsoleServer

Program jest uruchamiany na komputerze pokładowym gokarta. Po uruchomieniu dokonuje on cyklicznego odczytu danych z czujników, ich wstępnego przetwarzania oraz wysłania na modem radiowy. Ekran programu obrazuje rys. 3.6.



Rys. 3.6. Ekran programu *KnestConsoleServer*

Konfiguracja programu jest wczytywana z pliku *KnestTelemetry.cfg* znajdującego się w folderze *KnestLog*. Jeśli plik nie zostanie znaleziony, program wygeneruje go automatycznie oraz wczyta domyślne zawartości. Format pliku konfiguracji przedstawia rys. 3.7.

```
[ComPorts]
ACCELERATOR=COM1
GPS=COM2
ARDUINO=COM3
MODEM=COM4
IP=192.168.1.108
TCP=false
```

Rys. 3.7. Plik konfiguracji program *KnestConsoleServer*

Wszystkie dane przekazane do modemu, są również zapisywane do pliku dziennika w formacie tekstowym. Umożliwia to późniejszą analizę zgromadzonych danych oraz ułatwia znalezienie ewentualnych problemów w komunikacji z komputerem zdalnym. Format pliku przedstawia rys. 3.8.

Wymiana danych pomiędzy komputerem pokładowym gokart a komputerem zdalnym na którym odbywa się wizualizacja oraz składowanie danych, jest realizowana poprzez modemy radiowe. Standardem komunikacji jest interfejs RS232.

log_20151104_141525.txt — Notatnik

Plik Edycja Format Widok Pomoc

```
KNEST Telemetry started
14:15:25        Port detected: COM1
14:15:25        Port detected: COM2
14:15:25        Port detected: COM3
14:15:25        Port detected: COM4
14:15:25        MODEM port opening on COM4
14:15:25        MODEM port start writing thread
14:15:25        ACCEL port opening on COM1
14:15:25        ACCEL port start reading thread
14:15:26        #V#: $4,-4,231,47,17,-27,140,198,-401#
14:15:26        GPS port opening on COM2
14:15:27        GPS port start reading thread
14:15:28        Arduino port opening on COM3
14:15:28        #G#:$GPGGA,131530.000,0000.0000,N,00000.0000,E,0,00,0.0,0.0,M,0.0,M,,0000*68
14:15:28        Arduino port start reading thread
14:15:28        #G#:$GPGSA,A,1,,,,,,,,,,,,,0.0,0.0,0.0*30
14:15:28        #G#:$GPRMC,131530.000,V,0000.0000,N,00000.0000,E,000.0,000.0,041115,,,N*73
14:15:29        #V#: $3,-6,231,55,16,-31,115,203,-390#
14:15:29        #G#:$GPGGA,131531.000,0000.0000,N,00000.0000,E,0,00,0.0,0.0,M,0.0,M,,0000*69
14:15:29        #V#: $4,-5,231,50,14,-32,57,128,-365#
14:15:29        #G#:$GPGSA,A,1,,,,,,,,,,,,,0.0,0.0,0.0*30
14:15:29        #V#: $5,-6,230,53,16,-28,57,128,-365#
14:15:29        #G#:$GPRMC,131531.000,V,0000.0000,N,00000.0000,E,000.0,000.0,041115,,,N*72
14:15:29        #V#: $5,-7,231,50,14,-27,78,163,-371#
14:15:29        #G#:$GPVTG,000.0,T,,M,000.0,N,000.0,K,N*02
14:15:29        #V#: $6,-4,233,46,13,-28,92,155,-381#
14:15:30        #G#:$GPGGA,131532.000,0000.0000,N,00000.0000,E,0,00,0.0,0.0,M,0.0,M,,0000*6A
14:15:30        #V#: $3,-4,232,52,7,-30,37,143,-353#
14:15:30        #G#:$GPGSA,A,1,,,,,,,,,,,,,0.0,0.0,0.0*30
14:15:30        #V#: $3,-4,231,48,17,-30,134,185,-400#
14:15:30        #G#:$GPGSV,1,1,01,03,00,000,22*4B
14:15:30        #V#: $3,-5,231,52,11,-29,134,185,-400#
14:15:30        #G#:$GPRMC,131532.000,V,0000.0000,N,00000.0000,E,000.0,000.0,041115,,,N*71
```

Rys. 3.8. Plik dziennika programu *KnestConsoleServer*

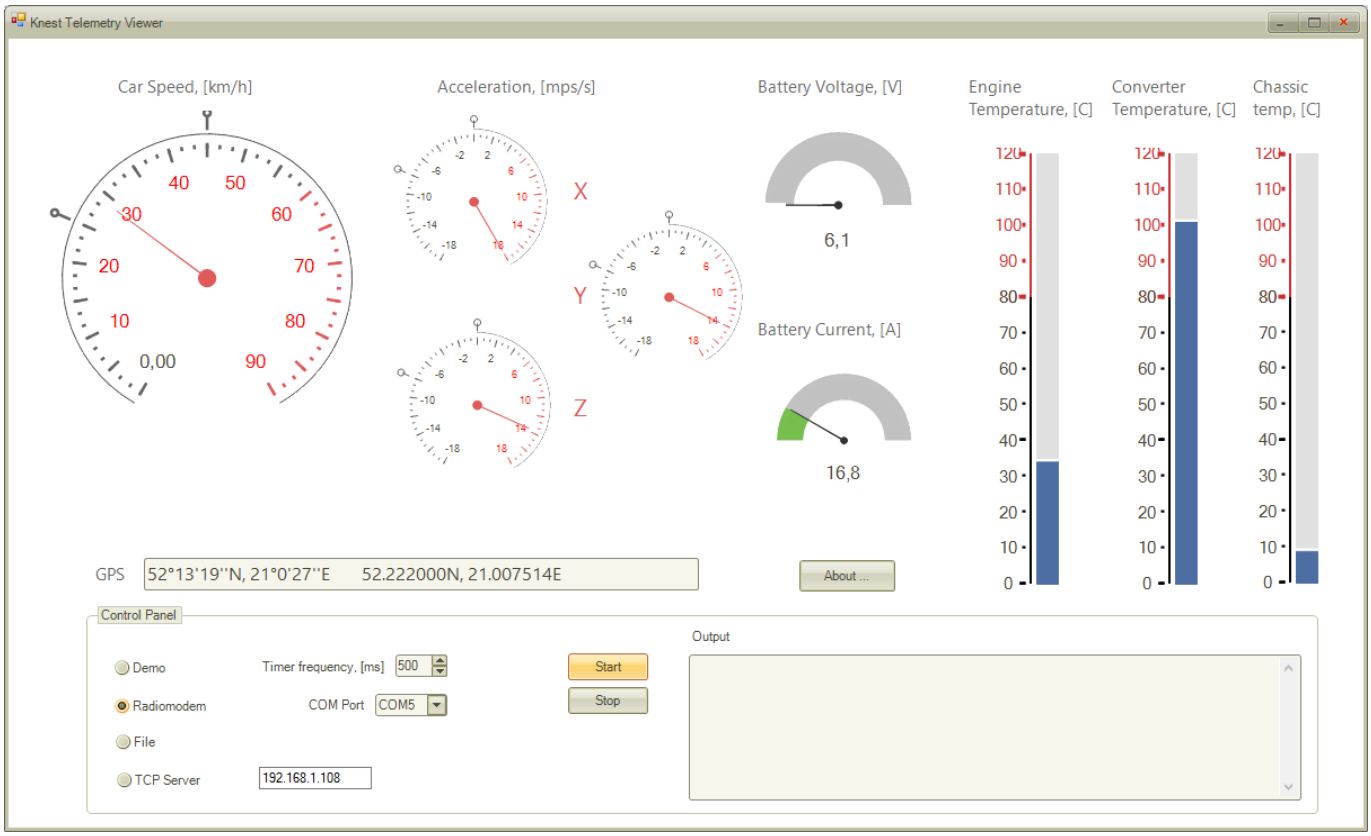
3.3. Program KnestTeleViewer

Program *KnestTeleViewer* dokonuje odczytu danych telemetrycznych z radiomodemu oraz przedstawia je w formie graficznej (rys. 3.9). Aplikacja typu *desktop* została zrealizowana z wykorzystaniem technologii *WinForms* platformy Microsoft .NET 4.0 i działa w systemach operacyjnych Windows 7/8/10 [20].

Program po uruchomieniu znajduje się w trybie oczekiwania. Uruchomienie odczytu odbywa się poprzez wciśnięcie przycisku [Start]. W celu demonstracji działania aplikacji bez dołączonego radiomodemu, lub w sytuacji kiedy dane z modemu nie są dostępne, został opracowany tryb demonstracyjny. Można go włączyć za pomocą przycisku [Demo] w panelu sterowania.

Przełączenie programu w tryb odczytu danych z radiomodemu odbywa się poprzez włączenie opcji [Telemetry] w panelu sterowania.

Program umożliwia również wizualizację danych zarejestrowanych oraz zapisanych w pliku LOG poprzez program *KnestConsoleServer*. W tym celu należy wybrać opcję przełącznika trybu [File], następnie za pomocą przycisku […] wybrać plik dziennika *.txt oraz uruchomić odtwarzanie za pomocą przycisku [||>]. Szybkość odtwarzania można regulować za pomocą opcji [Timer Frequency].



Rys. 3.9. Okno główne programu *KnestTeleViewer*

Kod źródłowy programu stanowi Załącznik 4.

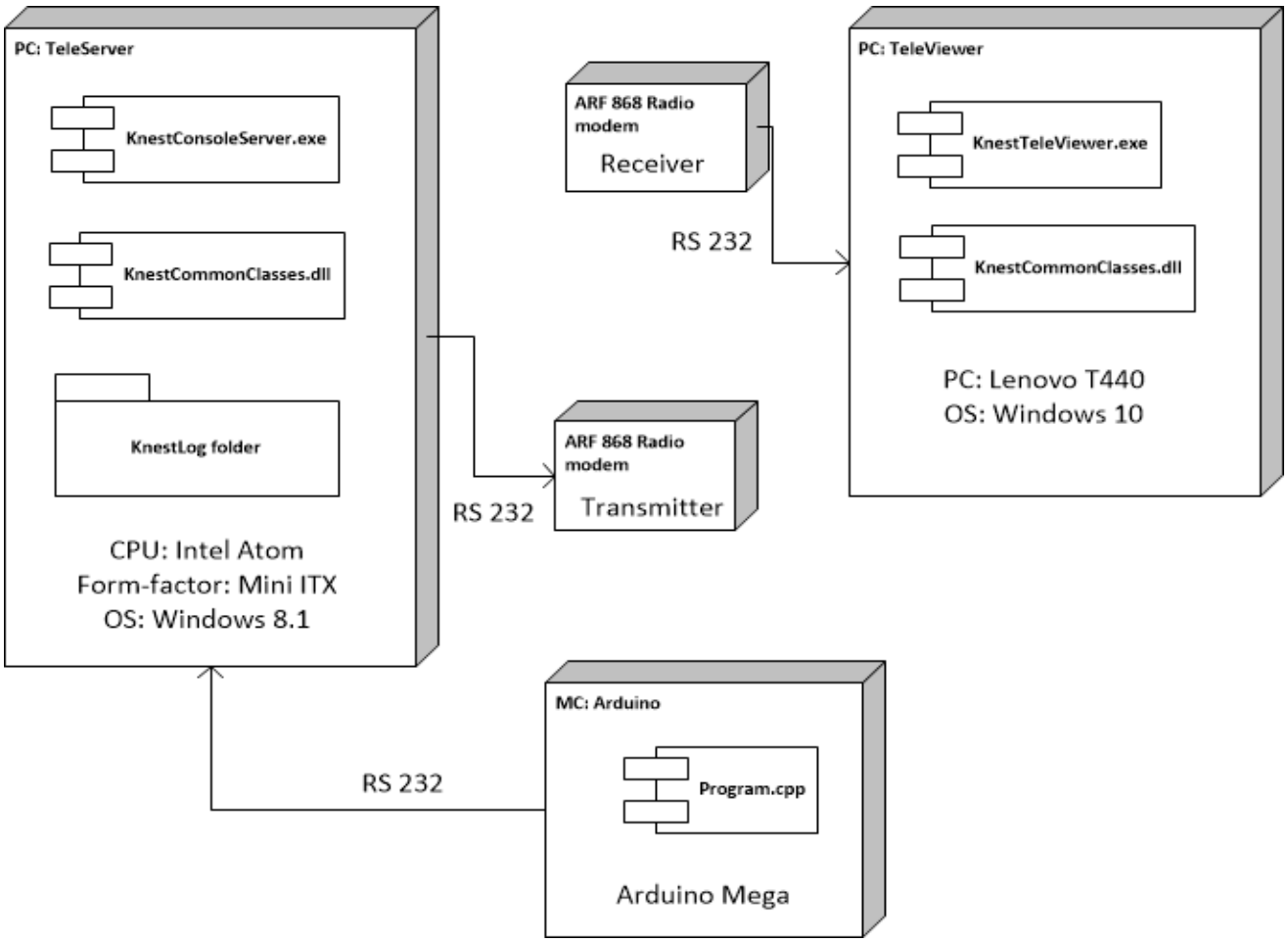
3.4. Konfiguracja sprzętowa systemu informatycznego telemetrii

Program do odczytu danych pomiarów z czytników jest zapisany w pamięci stałej płyty głównej kontrolera pomiarów zbudowanego na bazie Arduino. Dane odczytane są przekazywane do komputera pokładowego za pomocą złącza RS 232.

Na komputerze pokładowym działającym pod sterowaniem systemu operacyjnego *Windows Professional 8.1* zainstalowano oprogramowanie *KnestConsoleServer*. Zebrane oraz zinterpretowane dane pomiarów są doprowadzane do jednolitej postaci oraz przekazywane do modemu radiowego za pośrednictwem interfejsu RS232. Modem działa w trybie nadawania (*Transmitter*). Konfiguracja modemu odbywa się za pomocą oprogramowania dostarczonego przez producenta.

Po stronie odbiorcy dane zostają odczytane przez modem radiowy działający w trybie nasłuchiwania (*Receiver*). Następnie za pośrednictwem interfejsu RS 232 są one przekazywane do aplikacji *KnestTeleViewer* zainstalowanej na komputerze zdalnym (komputer wizualizacji). W projekcie wykorzystano w tym celu laptopa *Lenovo T440* z systemem operacyjnym *Windows 10*.

Na rys. 3.10. zostały przedstawione opisane wyżej komponenty fizyczne układu telemetrii wraz z zainstalowanym na nich oprogramowaniem.



Rys. 3.10. Diagram wdrożenia oraz komponentów fizycznych

Wnioski

Efektem końcowym projektu jest stworzenie systemu telemetrycznego w postaci urządzeń komputerowych oraz oprogramowania, przeznaczonego do elektronicznego pomiaru, rejestracji, przekazania na odległość oraz wizualizacji wybranych parametrów ruchu pojazdu elektrycznego: prędkości, przyspieszenia, temperatury, współrzędnych GPS, prądu oraz napięcia na akumulatorze.

W pierwszym etapie projektu zaproponowano koncepcję systemu składającego się z układu pomiarowego, komputera pokładowego umieszczonego w pojeździe elektrycznym, modemów radiowych przekazujących dane telemetryczne na odległość, oraz programu do wizualizacji wyników pomiarów. Opracowano schemat ideowy systemu w postaci graficznej. Określono parametry pojazdu podlegające monitorowaniu, sposób ich pomiaru oraz format zapisu. Zaproponowano układy elektroniczne do pomiarów każdego z monitorowanych parametrów.

W drugim etapie projektu zakupiono komponenty niezbędne do skonstruowania sprzętowej części systemu telemetrycznego, w tym płytę główną mikrokomputera pokładowego, czujniki i sensory pomiarowe, radiomodemy do komunikacji bezprzewodowej oraz inne podzespoły pomocnicze. Zgromadzono dokumentację dot. poszczególnych czujników oraz dokonano ich przetestowania poprzez podłączenie do komputera stacjonarnego, za pomocą programów dedykowanych dla każdego z urządzeń.

Etap trzeci projektu dotyczył prac konstrukcyjnych nad urządzeniem. W jego trakcie zaprojektowano w postaci schematu ideowego oraz fizycznie wykonano płytę kontrolera pomiarów napięcia, prądu oraz temperatury. Po wlutowaniu komponentów elektronicznych, płytkę kontrolera pomiarów połączono z układem scalonym na bazie mikrokontrolera Arduino. Sygnały z kontrolera pomiarów doprowadzono do minikomputera pokładowego za pośrednictwem portu RS232. Komputer pokładowy typu PC zbudowano na bazie platformy miniITX. Sygnały z akcelerometru oraz czujnika GPS podłączono bezpośrednio do portów RS232 na komputerze pokładowym. Po uruchomieniu zestawu za pomocą oprogramowania zainstalowanego na komputerze pokładowym skonfigurowano czujniki odczytu oraz dokonano ich kalibracji. Kontroler pomiarów zaprogramowano na odczyt oraz konwersję danych wejściowych zgodnie z przyjętymi jednostkami miary.

Etap czwarty projektu polegał na napisaniu szeregu aplikacji sterujących systemem. Program sterujący kontrolerem pomiarów został opracowany w języku C a następnie zapisany w pamięci stałej mikrokontrolera Arduino. Równolegle opracowano program konsolowy do odczytu danych z czujników, ich konsolidacji w pakiety oraz wysłania na radiomodem działający w trybie nadawcy. Program ten został zainstalowany na komputerze pokładowym w środowisku Windows. Przy programowaniu użyto języka C# oraz platformy .NET. Ostatecznie, opracowano program do odczytu danych z modemu radiowego działającego w trybie odbiorcy. Program posiada graficzny interfejs użytkownika GUI i pozwala na

monitorowanie bieżących wartości wszystkich parametrów, odczytywanych z radiomodemu. Opracowano projekt obudowy komputera pokładowego na bazie standardowej obudowy aluminiowej, dostosowanej do specyfiki projektu. Komputer pokładowy w obudowie umieszczono na gokarcie elektrycznym oraz podłączono do układów zasilania.

Końcowy piąty etap projektu obejmował testowanie systemu oraz napisanie dokumentacji sprawozdawczej. Testowanie systemu odbyło się na stanowiskach laboratoryjnych z wykorzystaniem aparatury badawczej, jaką dysponuje Wydział Transportu. Dokonano również testowania systemu w środowisku rzeczywistym, w warunkach terenowych. Podczas testowania stwierdzono zgodność efektów działania systemu z założeniami projektowymi. Po przeanalizowaniu wyników pomiarów zaproponowano również sposoby udoskonalenia systemu oraz nakreślono potencjalne ścieżki modyfikacji. Na podstawie wyników prac sporządzono sprawozdanie z projektu.

Wykonany projekt może być podstawą do prowadzenia dalszych prac badawczych w zakresie rozszerzenia liczby monitorowanych parametrów, wykorzystania bazy danych do gromadzenia wyników pomiarów, analizy wielowymiarowej zarejestrowanych danych oraz dostosowania urządzenia do potrzeb różnych kategorii pojazdów elektrycznych, w tym pojazdu elektrycznego dla osób niepełnosprawnych zbudowanego na Wydziale Transportu PW.

Planowana jest prezentacja wyników projektu na Konferencji Naukowej Transport XXI wieku. Wyniki projektu będą omówione na Seminarium Zakładowym. W planach jest również demonstracja efektów projektu podczas pikników naukowych oraz na innych imprezach promujących Wydział Transportu Politechniki Warszawskiej.

Budowa systemu telemetrii pozwoliła studentom zapoznać się z możliwościami współczesnych platform prototypowania na bazie mikrokontrolerów oraz specyfiką ich zastosowania w transporcie, zasadami projektowania złożonych systemów informatyczno-elektronicznych oraz podstawami ich programowania. Należy również wymienić nabyte umiejętności pracy zespołowej z wykorzystaniem współczesnej profesjonalnej platformy BaseCamp.

Materiały z realizacji projektu zostały zaprezentowane na stronie internetowej Koła Naukowego KNEST http://www.knest.pw.edu.pl.

Bibliografia

1. Moćko, W.; Wojciechowski, A.; Ornowski, M. Perspektywy rozwoju rynku samochodów elektrycznych w najbliższych latach. Transport Samochodowy, 2011

2. Barański, Marcin, et al. Diagnostyka i monitoring trakcyjnych maszyn elektrycznych. Logistyka, 3/2015.

3. Fryśkowski B., Grzejszczyk E., Systemy transmisji danych, Wydawnictwo Komunikacji i Łączności, Warszawa 2010.

4. Felser M.: The Fieldbus Standards: History and Structure. Technology Leadership Day, 2002, Organised by MICROSWISS Network, HTA Luzern, Oktober 2002.

5. Chaładyniak D., Wybrane technologie bezprzewodowej transmisji danych, Zeszyty Naukowe 87 – 101; Warszawska Wyższa Szkoła Informatyki, Warszawa 2015.

6. Michałek T.: Radiomodemy Satel. Pomiary Automatyka Robotyka, nr 9, 2003, 36-38

7. Wrona, R. , Ziółkowski, E.: Transmisja bezprzewodowa w zdalnych komputerowych systemach pomiarowo-sterujących. Archiwum Odlewnictwa 2002 | R. 2, nr 3 | 144--148

8. T. Parker, M. Sportach, TCP/IP: Księga eksperta. HELION, 2000.

9. P. Roshan, J. Leary, Bezprzewodowe sieci LAN 802.11. Podstawy, Mikom, 2004.

10. Gorczyca P.: Telemetria urządzeń z wykorzystaniem technologii GSM/GPRS, Probl. Transp. 2006 t. 1 z. 1, s. 161–166.

11. Januszewski, Jacek. Systemy satelitarne GPS, Galileo i inne. Wydawnictwo Naukowe PWN, 2006.

12. Margolis M.: Arduino Cookbook, O'Reilly Media, Inc. 2011.

13. Raspberry Pi Foundation - http://www.raspberrypi.org/

14. Platforma myRIO - http://www.ni.com/myrio/

15. Dokumentacja techniczna czujnika Razor IMU (http://botland.com.pl/czujniki-9dof-imu/2949-razor-imu-akcelerometr-zyroskop-i-magnetometr-9-stopni-swobody-sparkfun.html)

16. E. Dudek: Kwantowy efekt Halla i jego zastosowanie do odtwarzania jednostki miary oporu elektrycznego. Podstawowe Problemy Metrologii, Ustroń, maj 2005.

17. Dokumentacja techniczna płyty głównej Mitac PD12TI (http://www.mini-itx.com.pl/p/9/323/plyta-glowna-mitac-pd12ti-intel-d2500cc-mini-itx-plyty-glowne-itx.html)

18. Muc, Adam, et al. Zastosowanie platform cyfrowych Arduino i Raspberry Pi w nauczaniu sterowania obiektem pneumatycznym. Zeszyty Naukowe Wydziału Elektrotechniki i Automatyki Politechniki Gdańskiej, 2015

19. Dokumentacja techniczna radiomodemu ARF868 (http://www.adeunis-rf.com/en)

20. C.Nagel, B.Evjen, J.Glynn, K.Watson, M.Skinner. Professional C# 4.0 and .NET 4. Wrox, 2010

Załącznik 1. Kod źródłowy programu mikrokontrolera Arduino

```
#include <SimpleTimer.h>

float temperatura1;
float temperatura2;
float temperatura3;
float R;
float hass;
float v48;
float predkosc, predkosc0, predkosc2, predkosc3;
String dane;
boolean predkoscID = false;
boolean predkoscIDWysylka = false;
boolean tmp = false;
int timerDlaV;
unsigned long time1, time2, time;
SimpleTimer timerDlaWysylaniaDanych;

void setup() {
  Serial1.begin(9600);
  Serial.begin(9600);
  timerDlaWysylaniaDanych.setInterval(200, sendData);
  get_predkosc();
}

void loop() {
  get_temperature1();
  get_temperature2();
  get_temperature3();
  get_hass();
  get_v48();

  timerDlaWysylaniaDanych.run();
}

void get_temperature1(){
  temperatura1=analogRead(A4);
  R = ((10240000/temperatura1)-10000);

temperatura1=(1/(0.001129148+(0.000234125*(log(R)))+(0.0000000876741*(
log(R)*log(R)*log(R)))))-273.15;
}
void get_temperature2(){
  temperatura2=analogRead(A3);
  R = ((10240000/temperatura2)-10000);

temperatura2=(1/(0.001129148+(0.000234125*(log(R)))+(0.0000000876741*(
log(R)*log(R)*log(R)))))-273.15;
}
```

```
void get_temperature3(){
  temperatura3=analogRead(A2);
  R = ((10240000/temperatura3)-10000);

temperatura3=(1/(0.001129148+(0.000234125*(log(R)))+(0.0000000876741*(
log(R)*log(R)*log(R)))))-273.15;
}
void get_hass(){
  hass=analogRead(A1);
  hass = ((hass-515)*0.5818);
}
void get_v48(){
  v48=analogRead(A0);
  v48 = (((int(v48)*0.1)*55)/1024)*10;
}
void get_predkosc(){
  predkoscIDWysylka=false;
  attachInterrupt(digitalPinToInterrupt(2), get_predkosc2, HIGH);
}
void get_predkosc2(){
  if(predkoscID==false){
    predkoscID = true;
    time1=millis();
    detachInterrupt(digitalPinToInterrupt(2));
    get_predkosc();
  }
  else if(predkoscID==true){
    time2=millis();
    predkoscID = false;
    time=time2-time1;
    time1=0;
    time2=0;
    predkosc=(0.87/time*10)*36;
    //Serial.println();
    //Serial1.print("V:"+String(predkosc));
    time=0;
    detachInterrupt(digitalPinToInterrupt(2));
    get_predkosc();
  }
}

void sendData(){
    if(isinf(predkosc)){
      predkosc2=predkosc0;
    }else{
      if(predkosc==0 && predkosc0==0){
        predkosc2=predkosc0;
      }else{
        predkosc2=predkosc0;
        predkosc0=predkosc;
      }
```

```
    }
    if(predkosc2>=310){
      predkosc=0;
    }
    dane=String(String(temperatura1) + " " + String(temperatura2) + "
" + String(temperatura3) + " " + String(v48) + " " + String(predkosc2)
+" "+String(hass));
    Serial1.print(dane);
    predkosc=0;
}
```

Załącznik 2. Kod źródłowy klas biblioteki KnestCommonClasses

Plik AccelReader.cs

```
namespace KnestCommonClasses
{
    public class AccelReader : ComReader
    {
        public override string DecodeMessage(string msg)
        {
            if( !(msg.StartsWith("\r$") && msg.EndsWith("\r")) )
            {
                return string.Empty;
            }

            return "#V#: " + msg.Replace("\r", "").Replace("\n", "") + "\n";
        }
    }
}
```

Plik ArduinoReader.cs

```
using System;

namespace KnestCommonClasses
{
    public class ArduinoReader : ComReader
    {
        public override string DecodeMessage(string msg)
        {
            var s = msg.Split(' ');
            for(int i=1; i<s.Length; i++ )
            {
                if( i <= 3 )
                {
                    // temperatura
                    double s1 = double.Parse(s[i]);
                    double t = (5.0 * s1 * 100.0) / 1024.0;
                    s[i] = string.Format("T{0}={1}", i, t.ToString("##0.0"));
                }
            }

            msg = String.Empty;
            foreach (string x in s)
            {
                msg += x + " ";
            }

            return "#A#: " + msg;
        }

    }
}
```

Plik ComPort.cs

```
using System.IO.Ports;

namespace KnestCommonClasses
{
    public class ComPort
    {
        #region Events

        public delegate void DataReceivedHandler(string message);

        public event DataReceivedHandler DataReceived;

        #endregion

        public void SendMessage(string msg)
        {
            DataReceived?.Invoke(msg);
        }

        protected SerialPort _comPort;

        public virtual void Init(string com, int baudRate)
        {
            _comPort = new SerialPort();

            _comPort.PortName = com;
            _comPort.BaudRate = baudRate;
            _comPort.Parity = Parity.None;
            _comPort.DataBits = 8;
            _comPort.StopBits = StopBits.One;
            _comPort.Handshake = Handshake.None;
            _comPort.ReadTimeout = 1000;
            _comPort.WriteTimeout = 1000;
            _comPort.Open();
        }

        public void Write(string message)
        {
            if( _comPort == null || _comPort.IsOpen == false )
            {
                return;
            }

            _comPort.Write(message);
        }

        public void ClosePort()
        {
            if( _comPort != null && _comPort.IsOpen )
            {
                _comPort.Close();
            }
        }

    }

}
```

Plik ComReader.cs

```
using System;
using System.Threading;

namespace KnestCommonClasses
{
    public abstract class ComReader : ComPort
    {

        public void StartLoop()
        {
            try
            {
                Thread t = new Thread(ReadLoop);
                t.Start();
            }
            catch (Exception ex)
            {
                SendMessage("ERROR: " + ex.Message);
            }
        }

        public virtual void ReadLoop()
        {
            while (true)
            {
                Thread.Sleep(100);
                try
                {
                    string msg = _comPort.ReadLine();
                    msg = DecodeMessage(msg);
                    if ( msg.Length > 0 )
                    {
                        SendMessage(msg);
                    }

                }
                catch (Exception ex)
                {
                    SendMessage("ERR" + ex.Message);
                }
            }
        }

        public abstract string DecodeMessage(string msg);
    }
}
```

Plik Pomiar.cs

```
namespace KnestCommonClasses
{
    public class Pomiar
    {
        public float Speed { get; set; }                    // prędkość
        public float Acceleration { get; set; }             // przyspieszenie
        public float BatteryVoltage { get; set; }           // napięcie
        public float BatteryCurrent { get; set; }           // prąd
```

```
        public float EngineTemperature { get; set; }       // temperatura
        public float ConverterTemperature { get; set; }    // temperatura
        public Gps Coordinates { get; set;  }              // współrzędne GPS


        public Pomiar()
        {
            Speed = 0f;
            Acceleration = 0f;
            BatteryVoltage = 0f;
            BatteryCurrent = 0f;
            EngineTemperature = 0f;
            ConverterTemperature = 0f;

            Coordinates = new Gps(0,0);
        }

    }
}
```

Plik Gps.cs

```
using System;

namespace KnestCommonClasses
{
    [Serializable]
    public class Gps
    {
        private const double SrednicaZiemi = 12756.274;

        public double Lon { get; set; } // długość geograficzna (X)
        public double Lat { get; set; } // szerokość geograficzna (Y)

        public Gps(double lon, double lat)
        {
            Lon = lon;
            Lat = lat;
        }

        public override string ToString()
        {
            return "Lon=" + Lon.ToString("0.00000") + ",\nLat=" + Lat.ToString("0.00000");
        }

        // zwraca odległość pomiędzy dwoma punktami na mapie, km
        public static double GetDistanseKm(Gps pa, Gps pb)
        {
            var cosq =
                Math.Sin(pa.Lon)*Math.Sin(pb.Lon) +
                Math.Cos(pa.Lon)*Math.Cos(pb.Lon)*Math.Cos(pa.Lat - pb.Lat);

            var res = (2.0*Math.PI*(SrednicaZiemi/2)*Math.Acos(cosq)/360.0);

            return res;
        }

    }
}
```

Załącznik 3. Kod źródłowy programu KnestConsoleServer

```
using KnestCommonClasses;
using System;
using System.IO;
using System.IO.Ports;
using System.Runtime.CompilerServices;
using System.Threading;

namespace KnestConsoleServer
{
    class Program
    {
        #region Configuration

        private string ACCEL_COM = "COM1";
        private string GPS_COM = "COM2";
        private string ARDUINO_COM = "COM3";
        private string MODEM_COM = "COM4";

        #endregion

        #region Members

        private GpsReader _gpsReader;
        private AccelReader _accelReader;
        private ArduinoReader _arduinoReader;
        private RfModem _rfModem;

        private string _logFileName;
        private StreamWriter _logWriter;

        #endregion


        static void Main(string[] args)
        {
            new Program().Run();
        }

        [MethodImpl(MethodImplOptions.Synchronized)]
        private void AddMessage(string msg)
        {
            var s = DateTime.Now.ToString("HH:mm:ss\t") + msg;

            Console.WriteLine(s);

            if( _logFileName != null )
            {
                _logWriter = new StreamWriter(_logFileName, true);
                _logWriter.WriteLine(s);
                _logWriter.Close();
            }

            _rfModem?.Write(s);
        }

        private void Run()
        {
            AddMessage("KNEST TeleServer started");
            try
```

```
    {
        CreateLog();

        CreateObjects();
        WireEvents();
        EnumerateComPorts();

        StartModem();
        StartAccel();
        StartGps();
        StartArduino();

        Console.WriteLine("Press ESC to exit");
        ConsoleKey key;
        do
        {
            key = Console.ReadKey().Key;
            Thread.Sleep(500);
        }
        while (key != ConsoleKey.Escape);
    }
    catch(Exception ex)
    {
        AddMessage("ERROR: " + ex.Message);
    }
}

#region OnLoad

private void CreateObjects()
{
    _rfModem = new RfModem();
    _gpsReader = new GpsReader();
    _accelReader = new AccelReader();
    _arduinoReader = new ArduinoReader();

}

private void CreateLog()
{
    var folder = Path.Combine(
            Environment.GetFolderPath(Environment.SpecialFolder.Desktop), "KnestLog");
    if (!Directory.Exists(folder))
    {
        Directory.CreateDirectory(folder);
    }
    var name = "log_" + DateTime.Now.ToString("yyyyMMdd_HHmmss") + ".txt";
    _logFileName = Path.Combine(folder, name);

    _logWriter = new StreamWriter(_logFileName);
    _logWriter.WriteLine("KNEST Telemetry started");
    _logWriter.Close();
}

private void WireEvents()
{
    _gpsReader.DataReceived += OnSensorDataReceived;
    _accelReader.DataReceived += OnSensorDataReceived;
    _arduinoReader.DataReceived += OnSensorDataReceived;
}

private void EnumerateComPorts()
```

```
{
    string[] x = SerialPort.GetPortNames();
    foreach (var port in x)
    {
        AddMessage("Port detected: " + port);
    }
}

private void OnSensorDataReceived(string message)
{
    AddMessage(message);
}

#endregion

#region Sensors

private void StartGps()
{
    try {
        var com = GPS_COM;
        AddMessage("GPS port opening on " + com);
        _gpsReader.Init(com, 9600);
        AddMessage("GPS port start reading thread");
        _gpsReader.StartLoop();
    }
    catch(Exception ex)
    {
        AddMessage("GPS ERROR: " + ex.Message);
    }
 }

private void StartAccel()
{
    try
    {
        var com = ACCEL_COM;
        AddMessage("ACCEL port opening on " + com);
        _accelReader.Init(com, 57600);
        _accelReader.Write("4"); // skip device menu
        AddMessage("ACCEL port start reading thread");
        _accelReader.StartLoop();
    }
    catch(Exception ex)
    {
        AddMessage("ACCELERATOR ERROR: " + ex.Message);
    }
}

private void StartArduino()
{
    try
    {
        var com = ARDUINO_COM;
        AddMessage("Arduino port opening on " + com);
        _arduinoReader.Init(com, 9600);
        AddMessage("Arduino port start reading thread");
        _arduinoReader.StartLoop();
    }
    catch (Exception ex)
    {
        AddMessage("ARDUINO ERROR: " + ex.Message);
```

```
            }
        }

        #endregion

        #region Modem

        private void StartModem()
        {
            try
            {
                var com = MODEM_COM;
                AddMessage("MODEM port opening on " + com);
                _rfModem.Init(com, 9600);
                AddMessage("MODEM port start writing thread");
            }
            catch (Exception ex)
            {
                AddMessage("MODEM ERROR: " + ex.Message);
            }
        }

        #endregion
    }
}
```

Załacznik 4. Kod źródłowy programu KnestTeleViewer

Plik Program.cs

```
using System;
using System.Windows.Forms;

namespace KnestTeleViewer
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new MainFrame());
        }
    }
}
```

Plik PomiarController.cs

```
using System;
using System.Globalization;
using System.IO.Ports;
using System.Windows.Forms;
using KnestCommonClasses;

namespace KnestTeleViewer
{
    public class PomiarController
    {

        #region Events

        public delegate void DataReceivedEventHandler(Pomiar p);
        public event DataReceivedEventHandler DataReceived;

        #endregion

        #region Members

        private ESignalSource _signalSource;
        private readonly Timer _timer;
        private readonly Random _random;
        private SerialPort _modemPort;
        readonly Pomiar _currentPomiar;

        #endregion

        #region Constructor

        public PomiarController(ESignalSource source)
        {
            _timer = new Timer {Interval = 1000};
            _timer.Tick += OnTimerTick;
```

```
    _random = new Random();
    _currentPomiar = new Pomiar();

    InitModemPort("COM3", 9600);

    SwitchSource(source);
}

#endregion

#region DEMO

private void OnTimerTick(object sender, EventArgs e)
{

    var p = new Pomiar
    {
        Acceleration = (float) _random.NextDouble()*20,
        BatteryCurrent = (float) _random.NextDouble()*10,
        BatteryVoltage = (float) _random.NextDouble()*48,
        ConverterTemperature = (float) _random.NextDouble()*120,
        EngineTemperature = (float) _random.NextDouble()*120,
        Speed = (float)_random.NextDouble() * 90,
        Coordinates = new Gps( (int)(_random.NextDouble() * 360 - 180), (
                    int)(_random.NextDouble() * 180 - 90))
    };

    DataReceived?.Invoke(p);
}

#endregion

#region MODEM

private void InitModemPort(string com, int baudRate)
{
    _modemPort = new SerialPort
    {
        PortName = com,
        BaudRate = baudRate,
        Parity = Parity.None,
        DataBits = 7,
        StopBits = StopBits.One,
        Handshake = Handshake.None
    };

    _modemPort.DataReceived += OnModemDataReceived;
}

private void OnModemDataReceived(object sender, SerialDataReceivedEventArgs e)
{
    var msg = ((SerialPort)sender).ReadLine();
    UpdatePomiar(msg);
    DataReceived?.Invoke(_currentPomiar);
}

private void UpdatePomiar(string msg)
{
    if (!msg.Contains("\t")) return;
    if (msg.Split('\t').Length < 2) return;

    msg = msg.Split('\t')[1];
```

```
    if (msg.StartsWith("#V#"))
    {
        ParseVelocity(msg);
    }

    if (msg.StartsWith("#G#"))
    {
        ParseGps(msg);
    }

    if (msg.StartsWith("#A#"))
    {
        ParseArduino(msg);
    }

}


public Pomiar GetPomiar()
{
    return _currentPomiar;
}


#endregion

#region COMMON

public void SwitchSource(ESignalSource source)
{
    try
    {
        _signalSource = source;

        if (_signalSource == ESignalSource.Demo)
        {
            _timer.Start();
        }
        else
        {
            _timer.Stop();
        }

        if (_signalSource == ESignalSource.Modem)
        {
            if (_modemPort.IsOpen == false)
                _modemPort.Open();

        }
        else
        {
            if (_modemPort.IsOpen)
            {
                _modemPort.Close();
            }
        }

    } catch(Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
```

```
        }


        private string FormatDouble(double d, int decimals = 2)
        {
            return Math.Round(d, decimals).ToString(CultureInfo.InvariantCulture);
        }

        #endregion

    }
}
```

Plik MainFrame.cs
(wydruk nie zawiera kodu generowanego automatycznie, p. załącznik CD)

```
using System;
using System.Drawing;
using KnestCommonClasses;

namespace KnestTeleViewer
{
    public partial class MainFrame : RadForm
    {
        #region Members

        private PomiarController _pController;

        #endregion

        #region Properties

        public ESignalSource SignalSource
        {
            get
            {
                if( SignalSourceNoneRadio.IsChecked ) return ESignalSource.None;
                return SignalSourceDemoRadio.IsChecked ?
                            ESignalSource.Demo : ESignalSource.Modem;
            }

            set
            {
                switch (value)
                {
                    case ESignalSource.None:
                        SignalSourceNoneRadio.IsChecked = true;
                        break;
                    case ESignalSource.Demo:
                        SignalSourceDemoRadio.IsChecked = true;
                        break;
                    case ESignalSource.Modem:
                        SignalSourceModemRadio.IsChecked = true;
                        break;
                }
            }
        }

        #endregion
```

```
        #region Constructor

        public MainFrame()
        {
            InitializeComponent();
        }

        #endregion

        #region Startup

        private void MeLoad(object sender, EventArgs e)
        {
            _pController = new PomiarController(ESignalSource.None);
            WireEvents();
        }

        private void WireEvents()
        {
            CurrentGauge.ValueChanged += CurrentGaugeChanged;
            VoltageGauge.ValueChanged += VoltageGaugeChanged;
            SpeedGauge.ValueChanged += SpeedGaugeChanged;

            SignalSourceNoneRadio.ToggleStateChanged += OnSignalSourceChanged;
            SignalSourceDemoRadio.ToggleStateChanged += OnSignalSourceChanged;
            SignalSourceModemRadio.ToggleStateChanged += OnSignalSourceChanged;

            _pController.DataReceived += OnControllerDataReceived;
        }


        #endregion

        #region Visualisation

        private void ApplyValueToRadialGauge(RadRadialGauge radialGauge, ref float step)
        {
            if (radialGauge.Value + step > radialGauge.RangeEnd
                    || radialGauge.Value + step < radialGauge.RangeStart)
            {
                step = -step;
            }

            var aps =
new AnimatedPropertySetting(RadRadialGaugeElement.ValueProperty, radialGauge.Value,
                radialGauge.Value + step, 12, 40)
            { ApplyEasingType = RadEasingType.OutBounce };

            aps.ApplyValue(radialGauge.GaugeElement);
        }

        private void SetValueToRadialGauge(RadRadialGauge radialGauge, float value)
        {
            var aps =
                new AnimatedPropertySetting(RadRadialGaugeElement.ValueProperty,
radialGauge.Value, value, 12, 40)
                    { ApplyEasingType = RadEasingType.OutBounce };

            aps.ApplyValue(radialGauge.GaugeElement);
        }
```

```
void SpeedGaugeChanged(object sender, EventArgs e)
{
    if (SpeedGauge.Value >= 6f)
    {
        radialGaugeNeedle2.BackColor = Color.FromArgb(224, 90, 90);
        radialGaugeNeedle2.BackColor2 = Color.FromArgb(224, 90, 90);
    }
    else
    {
        radialGaugeNeedle2.BackColor = Color.Black;
        radialGaugeNeedle2.BackColor2 = Color.Black;
    }
}

void VoltageGaugeChanged(object sender, EventArgs e)
{
    if ( VoltageGauge.Value >= 50)
    {
        radialGaugeArc3.BackColor = Color.FromArgb(119, 190, 79);
        radialGaugeArc3.BackColor2 = Color.FromArgb(121, 191, 80);
    }
    else
    {
        radialGaugeArc3.BackColor = Color.FromArgb(224, 70, 71);
        radialGaugeArc3.BackColor2 = Color.FromArgb(224, 70, 71);
    }
}

void CurrentGaugeChanged(object sender, EventArgs e)
{
    if (CurrentGauge.Value < 50)
    {
        radialGaugeArc5.BackColor = Color.FromArgb(119, 190, 79);
        radialGaugeArc5.BackColor2 = Color.FromArgb(121, 191, 80);
    }
    else
    {
        radialGaugeArc5.BackColor = Color.FromArgb(224, 70, 71);
        radialGaugeArc5.BackColor2 = Color.FromArgb(224, 70, 71);
    }
}

#endregion

private void OnSignalSourceChanged(object sender, StateChangedEventArgs args)
{
    _pController.SwitchSource(SignalSource);
}

private void OnControllerDataReceived(Pomiar p)
{
    SetValueToRadialGauge(SpeedGauge, p.Speed);
    SetValueToRadialGauge(AccelGauge, p.Acceleration);
    SetValueToRadialGauge(VoltageGauge, p.BatteryVoltage);
    SetValueToRadialGauge(CurrentGauge, p.BatteryCurrent);

    EngineTempGauge.Value = p.EngineTemperature;
    ConverterTempGauge.Value = p.ConverterTemperature;
}

private void DatabaseSwitch_ValueChanged(object sender, EventArgs e)
{
```

```
            if (DatabaseSwitch.Value)
            {
                if (SqlServer.Connect() == false)
                {
                    DatabaseSwitch.Value = false;
                }
            }
            else
            {
                DatabaseSwitch.Value = false;
            }
        }
    }
}
```