POLITECHNIKA POZNAŃSKA

WYDZIAŁ AUTOMATYKI, ROBOTYKI I ELEKTROTECHNIKI
Instytut Robotyki i Inteligencji Maszynowej

Kierunek: Automatyka i Robotyka

Praca dyplomowa licencjacka

OPRACOWANIE ALGORYTMU WYKRYWANIA PARAMETRÓW LOTU RAKIETY NA PODSTAWIE ODCZYTÓW BAROMETRU

Maciej Kowalski, 151080

Promotor
dr hab. inż. Tomasz Pajchrowski

POZNAŃ 2025

Spis treści

Rozdział 1

Streszczenie

Poniższa praca przedstawia proces tworzenia algorytmu wykrywającego parametry lotu rakiety na podstawie odczytów barometru. Analiza wymagań postawionych przed algorytmem doprowadziła do zastosowania filtru Kalmana jako głównego komponentu filtrującego oraz pozwalającego na estymację prawdziwych parametrów lotu. Następnie przedstawiony został przebieg projektowania oraz testowania samego filtra, najpierw wpostaci liniowej (Linear Kalman Filter) a następnie w postaci rozszerzonej (Extended Kalman Filter), co pozwoliło osiągnąć wymaganą wydajność algorytmu. Opisana została również metoda określenia apogeum oraz innych faz lotu rakiety. Istotnym elementem pozwalającym na testowanie tworzonego algorytmu stał się program symulujący błędy pomiarowe uzyskiwane w trakcie fazy lotu z prędkością zbliżoną i wyższą niż prędkość jednego macha, pojawiających się z powodu przekraczania bariery dźwięku.
Opisany został również proces modelowania wysokości lotu rakiety z wykorzystaniem wskazań akcelerometru poprzez opisanie równań stanu a następnie doprowadzenie do postaci pożądanej przez filtr, a także sposób uzyskania poprawnych nastaw algorytmu.
W ostatnim etapie pracy zaprezentowana została analiza wydajności ostatecznej wersji algorytmu, propozycja implementacji w językach C i FreeRTOS oraz tendencje rozwojowe, które dają możliwość rozszerzenia funkcjonalności algorytmu do wykrywania faz lotu rakiety takich jak zapłon silnika, przyspieszanie, wypalenie silnika lub wybieg.

Rozdział 2

Abstract

This thesis presents the process of creating an algorithm for detecting rocket flight parameters based on barometer readings. The analysis of the requirements for the algorithm led to the use of the Kalman filter as the main filtering component, enabling the estimation of true flight parameters. The design and testing process of the filter is subsequently described, initially as a linear model (Linear Kalman Filter) and later as an extended model (Extended Kalman Filter), which allowed achieving the required algorithm performance. Additionally, the method for determining the apogee and other phases of rocket flight is explained. A significant component enabling the testing of the developed algorithm was a program simulating measurement errors observed during the flight phase at speeds near or exceeding the speed of sound, caused by breaking the sound barrier.
The process of modeling rocket flight altitude using accelerometer readings is also detailed by describing the state equations and subsequently adapting them to the form required by the filter, as well as the method for obtaining correct algorithm settings.
In the final stage of the work, the performance analysis of the final version of the algorithm is presented, along with a proposal for implementation in C and FreeRTOS, and development trends that offer the possibility of extending the algorithm's functionality to detect rocket flight phases such as engine ignition, acceleration, engine burnout, or coasting.

Rozdział 3

Spis symboli

Poniżej przedstawiony został spis symboli, skrótów i oznaczeń występujących w pracy:

- **LKF** - Linear Kalman Filter (z ang.) liniowy algorytm filtru Kalmana
- **EKF** - Extended Kalman Filter (z ang.) algorytm filtru Kalmana uwzględniający nieliniowość modelu lub równań pozwalających na przeliczenie pomiaru na zmienną stanu
- **F** - macierz tranzycji stanu
- **G** - macierz tranzycji wejścia
- **H** - macierz obserwatora stanu
- **K** - macierz wzmocnienia Kalmana
- **P** - macierz kowariancji określająca niepewność estymacji zmiennych stanu
- **R** - macierz kowariancji określająca niepewność pomiarową
- **Q** - macierz kowariancji określająca niepewność procesu obliczania modelu
- **CATS** - skrót od *Control and Telemetry Systems* - komercyjny komputer pokładowy przeznaczony do wykorzystania w amatorskich systemach rakietowych

Rozdział 4

Wstęp

Temat stworzenia algorytmu wykrywania parametrów lotu rakiety za pomocą odczytów barometru wynika z praktycznej potrzeby stworzenia takiego systemu w ramach koła naukowego Politechniki Poznańskiej PUT Rocketlab, które w bieżącym roku koncentruje swoje wysiłki na stworzeniu kolejnej wersji rakiety HEXA będącej flagowym projektem przygotowywanym na zbliżające się letnie zawody amatorskich zespołów rakietowych Spaceport America Cup 2025 odbywające się w Stanach Zjednoczonych. Celem projektu jest stworzenie algorytmu pozwalającego na określenie momentu osiągnięcia apogeum wysokości lotu rakiety, aby wskazać odpowiedni moment na wystrzelenie spadochronów pozwalających na sprowadzenie maszyny na powierzchnię ziemi bez uszkodzenia spowodowanego zbyt szybkim zderzeniem z ziemią. Do osiągnięcia tego celu wykorzystane zostaną przede wszystkim wskazania barometru oraz akcelerometru. Algorytm ten będzie elementem większego systemu pozwalającego na jak najdokładniejsze określenie momentu apogeum, składającego się dodatkowo z algorytmów opartych o wskazania akcelerometru oraz magnetometru, które nie są częścią tej pracy.
Zakres pracy określony został w następujących zadaniach projektowych:

1. przygotowanie teoretyczne - analiza komercyjnych algorytmów na licencji open source (np. CATS, Raven), lektura materiałów o filtrze kalmana (np. "Kalman Filter from the ground up", autor: Alex Becker), wsparcie zdobytej wiedzy dyskusjami z forum www.rocketryforum.com

2. wstępne przygotowanie algorytmu od strony matematycznej w języku Python i/lub środowisku Matlab na podstawie danych symulacyjnych wygenerowanych z programu OpenRocket na podstawie przygotowanej przez członków koła naukowego symulacji rakiety

3. implementacja w języku C oraz bibliotece FreeRTOS algorytmu na stworzonym przez koło naukowe komputerze pokładowym rakiety

4. testy końcowe na podstawie danych z symulacji OpenRocket - sprawdzenie poprawności działania, implementacji oraz integracji z pozostałymi komponentami komputera pokładowego

Praca została oparta przede wszystkim na podręczniku *„Kalman Filter from the Ground up"* napisaną przez Alexa Beckera, w której od podstaw przedstawiony został filtr Kalmana razem z wyprowadzeniem wszystkich niezbędnych wzorów i przedstawieniem podstaw matematycznych, koniecznych do zrozumienia tego tematu. Drugim źródłem pomocnym przy tworzeniu założeń algorytmu jest kod źródłowy oraz instrukcja użytkowania komputera pokładowego CATS przeznaczonego do amatorskich zestawów rakietowych. Oba materiały znajdują się pod licencją GNU General Public License v3.0, która umożliwia min. modyfikację, dystrybucję oraz wykorzystanie patentowe. Instrukcja opisuje ogólną strukturę algorytmu oraz wyprowadzenie matematyczne,

natomiast dzięki dołączonym do niej skryptom możliwe było zapoznanie się z praktyczną implementacją programu w języku C. Dużym wsparciem praktycznym przy pisaniu pracy były również wybrane wątki zamieszczone w forum internetowym www.rocketryforum.com.

W pierwszym rozdziale pracy autor przedstawił założenia teoretyczne wykorzystane w pracy, w szczególności równania składające się na liniowy i rozszerzony filtr Kalmana oraz model matematyczny opisujący lot rakiety. Ponadto zostały opisane fazy lotu rakiety, metody wykrycia apogeum oraz tematyka błędów pomiarowych w trakcie przekraczania bariery dźwięku i lotu z prędkością supersoniczną.
W drugim rozdziale pracy ukazano proces opracowania algorytmu, najpierw w oparciu o implementację liniowego filtru Kalmana i symulacji błędów w fazie transoniki z zastosowaniem rozkładu Gaussa, a następnie z użyciem rozszerzonego filtru Kalmana i symulując błąd o rozkładzie jednorodnym. Na koniec każdego z podrozdziałów przedstawiono wyniki, analizę i ewaluację algorytmów. W ostatnim podrozdziale ukazana została propozycja implementacji algorytmu w języku C i FreeRTOS.
W trzecim rozdziale przedstawione zostało podsumowanie pracy i wnioski końcowe.

Rozdział 5

Podstawy teoretyczne

5.1 Lot Rakiety

Rakieta koła naukowego PUT Rocketlab typu Hexa jest amatorską rakietą hybrydową, przeznaczoną do startu w kategoriach lotów na wysokość ok. 20 000 stóp, czyli w przybliżeniu 6 km. Prawidłowy lot takiej rakiety można podzielić na kilka podstawowych faz [2]:

- przygotowanie przed lotem - w trakcie tej fazy odbywa się kalibracja i przygotowanie systemów, rakieta znajduje się na stanowisku startowym
- wystrzał - następuje zapłon mieszanki paliwowej, po czym rakieta rozpoczyna lot z dużym, dodatnim przyspieszeniem w kierunku osi natarcia, zwrócona w stronę przestrzeni powietrznej
- wypalenie silnika - moment, w którym następuje koniec spalania paliwa a ciąg silnika zanika
- wybieg - faza lotu, w trakcie której maszyna leci w stronę przestrzeni kosmicznej siłą rozpędu nabytego w trakcie czasu działania silnika, pojazd jest również pod wpływem przyspieszenia grawitacyjnego, które powoduje samoczynną utratę prędkości rakiety względem ziemi
- apogeum - moment lotu, w którym rakieta osiąga najwyższą możliwą wysokość, natomiast prędkość w osi wystrzału jest równa zero, oś natarcia rakiety przechyla się do poziomu równoległego z powierzchnią ziemi, jest to moment aby algorytm wysłał sygnał inicjujący wystrzał spadochronów
- spadek swobodny - jeśli system odzyskiwania rakiety zadziałał poprawnie, następuje powolne opadanie w kierunku powierzchni ziemi
- kontakt z ziemią - moment, w którym maszyna styka się z powierzchnią, następuje zakończenie ostatniej fazy lotu

Celem postawionym przed algorytmem jest doprowadzenie do jak najdokładniejszego wskazania momentu osiągnięcia apogeum przez rakietę na podstawie pomiarów barometru. Aby to osiągnąć, konieczne jest określenie prędkości chwilowej rakiety z uwzględnieniem faktu, że zestaw czujników udostępniony do wykonania zadania nie obejmuje bezpośredniego pomiaru tej wartości. Niezbędne jest więc zastosowanie algorytmu estymującego prędkość rakiety na podstawie informacji o wysokości lotu względem ziemi wyliczonych z wskazań barometru. Nie można również pominąć błędów pomiarowych, które mogą doprowadzić do niewykonania zadania z powodu zbyt niedokładnej estymacji. Po uwzględnieniu wszystkich powyższych aspektów, jako podstawę algorytmu wykorzystano Filtr Kalmana.

5.2 Filtr Kalmana

Jest to algorytm pozwalający na estymację zmiennych stanu obiektów dyskretnych z wykorzystaniem pomiarów obarczonych niezerowym błędem. Polega on na połączeniu wartości obliczonych a priori z pomocą modelu oraz pomiarów z danej chwili, które następnie przeliczane są w jedną wartość za pomocą tzw. wzmocnienia Kalmana [14][1]. Dzięki temu istnieje możliwość wyznaczenia niedostępnych zmiennych stanu jedynie na podstawie bieżących, dostępnych wartości pomiarowych oraz znajomości modelu matematycznego. Filtr ten jest często stosowany w układach sensorycznych np. w robotyce, samolotach, pojazdach autonomicznych, radarach lub w innych systemach, które wymagają estymacji położenia lub innych wartości [9].
Oryginalnie filtr ten został wymyślony przez R.E. Kalmana w 1960 roku i opublikowany w referacie naukowym na temat rekurencyjnego rozwiązania problemu liniowego filtrowania danych dyskretnych, który zdobył szerokie uznanie min. w gronie naukowców z Ames Research Center for NASA w Mountain View w Kalifornii [1].

Filtr Kalmana można przedstawić w postaci trzech osobnych algorytmów opartych na tej samej zasadzie obliczeń, ale z pewnymi modyfikacjami pozwalającymi na różne zastosowania: [1]:

- Linear Kalman Filter (LKF) - algorytm wykorzystywany przy estymowaniu zmiennych stanu liniowych, czasowo-inwariantnych systemów i gdy wartości mierzone obarczone są błędem o rozkładzie Gaussa. W takim układzie filtr ma możliwość stać się filtrem optymalnym.
- Extended Kalman Filter (EKF) - algorytm wykorzystywany, gdy rozkład błędu pomiarowego nie jest rozkładem Gaussa lub gdy model obiektu jest nieliniowy. Poprzez odpowiednią modyfikację niektórych ze wzorów LKF i zastosowanie analitycznej linearyzacji modelu, filtr ten nadaje się do tego zastosowania, jednak nie będzie możliwe osiągnięcie optymalności.
- Unscented Kalman Filter (UKF) - filtr oferujący alternatywne podejście do nieliniowych systemów poprzez wykorzystanie linearyzacji statystycznej z zastosowaniem zestawu zasad. Ze względu na to, że w trakcie procesu opracowania algorytmu w niniejszej pracy zastosowane zostały tylko filtry LKF i EKF, ten temat nie zostanie dalej rozwinięty.

5.2.1 Liniowy filtr Kalmana (LKF) [1]

Liniowy filtr Kalmana składa się z pięciu równań podzielonych na dwie grupy - predykcji i aktualizacji. Składowymi tych równań są następujące macierze:

- F - macierz tranzycji stanu
- G - macierz tranzycji wejścia
- H - macierz obserwatora stanu
- K_n - macierz wzmocnienia Kalmana w chwili n
- P_n - macierz kowariancji określająca niepewność estymacji zmiennych stanu w chwili n
- R_n - macierz kowariancji określająca niepewność pomiarową w chwili n
- Q - macierz kowariancji określająca niepewność procesu obliczania modelu

Równania predykcji przedstawione poniżej pozwalają na przewidzenie wartości zmiennych stanu oraz macierzy kowariancji w kolejnej, dyskretnej chwili.

- Równanie ekstrapolacji stanu

$$\hat{x}_{n+1,n} = F\hat{x}_{n,n} + Gu_n \tag{5.1}$$

gdzie $\hat{x}_{n+1,n}$ to ekstrapolowana wartość wektora stanu w chwili $n+1$, $\hat{x}_{n,n}$ to wektor stanu w chwili n, natomiast u_n to wektor wejść systemu.

- równanie ekstrapolacji kowariancji

$$P_{n+1,n} = FP_{n,n}F^T + Q \tag{5.2}$$

gdzie $P_{n+1,n}$ to ekstrapolowana wartość macierzy kowariancji w chwili $n+1$, natomiast $P_{n,n}$ to wartość tej macierzy w chwili n.

Równanie 5.1 określane jest mianem równania ekstrapolacji stanu i pozwala na estymację wartości zmiennych wektora stanu w kolejnej dyskretnej próbce. Drugim jest równanie ekstrapolacji kowariancji oznaczone numerem 5.2. Dzięki temu równaniu istnieje możliwość estymacji niepewności zmiennych wektora stanu w następnej iteracji obliczeń.

Równaniami, które pozwalają na estymację aktualnych wartości zmiennych wektora stanu są równania aktualizacji.

- Równanie wzmocnienia Kalmana

$$K_n = P_{n,n-1}H^T \times (HP_{n,n-1}H^T + R_n)^{-1} \tag{5.3}$$

gdzie K_n to wzmocnienie Kalmana, natomiast $P_{n,n-1}$ to macierz kowariancji predyktowana w równaniu 5.2.

- Równanie aktualizacji stanu

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n \times (z_n - H\hat{x}_{n,n-1}) \tag{5.4}$$

gdzie $\hat{x}_{n,n}$ oznacza wartości wektora stanu w chwili n, $\hat{x}_{n,n-1}$ oznacza wartości wektora stanu predyktowane w równaniu 5.1, K_n to wzmocnienie Kalmana obliczone w równaniu 5.1, natomiast z_n to wartość pomiaru dokonanego w chwili n.

- Równanie aktualizacji kowariancji

$$P_{n,n} = (I - K_nH)P_{n,n-1} \times (I - K_nH)^T + K_nR_nK_n^T \tag{5.5}$$

gdzie $P_{n,n}$ to macierz kowariancji w chwili n, I to macierz jednostkowa, K_n to wzmocnienie Kalmana obliczone we wzorze 5.3 a $P_{n,n-1}$ to macierz kowariancji obliczona w równaniu 5.2.

Istnienie tych pięciu równań umożliwia działanie filtru Kalmana. Z racji tego, że niektóre działania są od siebie zależne, obliczane są one zgodnie z następującym cyklem:

1. Obliczenie wzmocnienia Kalmana
2. Aktualizacja stanu
3. Aktualizacja macierzy kowariancji
4. Predykcja równania stanu dla następnej iteracji

5. Predykcja macierzy kowariancji dla następnej iteracji

Wzmocnienie Kalmana jest współczynnikiem wagowym pozwalającym na określenie tego, czy filtr powinien bardziej ufać wartością mierzonym czy wyliczonym z modelu. Obliczane jest na podstawie predyktowanej w poprzedniej iteracji wartości macierzy kowariancji i może osiągnąć wartość od 0 do 1. W równaniu tym uwzględnia się również macierz niepewności pomiarowej R_n. Równanie aktualizacji stanu wykorzystuje obliczone uprzednio wzmocnienie Kalmana aby z odpowiednią wagą połączyć predyktowaną wartość zmiennych wektora stanu oraz pomiar dokonany w danej chwili. Następnie, na podstawie wzmocnienia Kalmana oraz uprzednio predyktowanej wartości macierzy P, dokonuje się aktualizacji macierzy kowariancji, również uwzględniając błąd pomiarowy. W celu sporządzenia wykresu estymowanych wartości lub wykorzystania wyniku filtru należy wziąć tę właśnie zaktualizowaną wartość wektora x
Po równaniach aktualizacji dokonuje się predykcji macierzy zmiennych stanu x poprzez wyliczenie wartości modelu oraz aktualizację macierzy kowariancji P z uwzględnieniem kowariancji procesu.

5.2.2 Rozszerzony filtr Kalmana (EKF) [1]

Algorytm rozszerzonego filtra Kalmana cechuje się dużym podobieństwem do jego wersji liniowej, jednak z uwzględnieniem problemu nieliniowości modelu lub nieliniowej relacji stanu do pomiaru, w którym nie występuje rozkład błędu w postaci Gaussa. W przypadku nieliniowego modelu dokonuje się analitycznej linearyzacji modelu w każdej kolejnej iteracji obliczeń. W praktyce dokonuje się tego poprzez mnożenie nieliniowej macierzy kowariancji z Jakobianami funkcji tranzycji:

$$P_{out} = \frac{\partial f}{\partial x} P_{in} \left(\frac{\partial f}{\partial x} \right)^T \tag{5.6}$$

gdzie P_{out} to macierz kowariancji ze zlinearyzowaną niepewnością, P_{in} to macierz kowariancji przed linearyzacją, natomiast $\frac{\partial f}{\partial x}$ to jakobian wektora stanu.
Podobnie wygląda linearyzacja w przypadku nieliniowej relacji stanu do pomiaru, gdzie wykorzystuje się Jakobian macierzy obserwatora stanu:

$$P_{out} = \frac{\partial h}{\partial x} P_{in} \left(\frac{\partial h}{\partial x} \right)^T \tag{5.7}$$

gdzie $\frac{\partial h}{\partial x}$ to jakobian funkcji przeliczającej wartość mierzoną na zmienną wektora stanu.

Z racji tego, że macierze tranzycji i obserwatora zawierają funkcje nieliniowe, oprócz linearyzacji macierzy kowariancji zmianie ulegają również równania filtru Kalmana. Po aktualizacji mają one następującą postać:

- Równanie ekstrapolacji stanu

$$\hat{x}_{n+1,n} = f(\hat{x}_{n,n}) + Gu_n \tag{5.8}$$

- równanie ekstrapolacji kowariancji

$$P_{n+1,n} = \frac{\partial f}{\partial x} P_{n,n} \left(\frac{\partial f}{\partial x} \right)^T + Q \tag{5.9}$$

- Równanie wzmocnienia Kalmana

$$K_n = P_{n,n-1}\left(\frac{\partial h}{\partial x}\right)^T \times \left(\frac{\partial h}{\partial x}P_{n,n-1}\left(\frac{\partial h}{\partial x}\right)^T + R_n\right)^{-1} \tag{5.10}$$

- Równanie aktualizacji stanu

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n \times (z_n - h(\hat{x}_{n,n-1})) \tag{5.11}$$

- Równanie aktualizacji kowariancji

$$P_{n,n} = (I - K_n\frac{\partial h}{\partial x})P_{n,n-1} \times (I - K_n\frac{\partial h}{\partial x})^T + K_n R_n K_n^T \tag{5.12}$$

gdzie na czerwono zostały oznaczone elementy różniące się od równań LKF, $f(\hat{x}_{n,n})$ to nieliniowa macierz tranzycji, $\frac{\partial f}{\partial x}$ to Jakobian macierzy tranzycji, $h(\hat{x}_{n,n-1})$ to nieliniowa funkcja obserwatora, natomiast $\frac{\partial h}{\partial x}$ to Jakobian funkcji obserwatora.

5.2.3 Wartości macierzy [1]

W celu wykonania obliczeń niezbędna jest również poprawne zdefiniowanie wartości macierzy wchodzących w ich skład. W celu obliczenia macierzy tranzycji stanu i tranzycji wejścia należy stworzyć model obiektu, a następnie rozwiązać równanie różniczkowe. Dla ekstrapolatora zerowego rzędu ogólne rozwiązanie równania 5.13:

$$\dot{x}(t) = Fx(t) + Gu(t) \tag{5.13}$$

Będzie posiadało formę przedstawioną w równaniu 5.14:

$$x(t + \Delta t) = e^{A\Delta t}x(t) + \int_0^{\Delta t} e^{A\Delta t}dtBu(t) \tag{5.14}$$

Co umożliwia wyprowadzenie następujących wzorów:

$$\begin{aligned} F &= e^{A\Delta t} \\ G &= \int_0^{\Delta t} e^{A\Delta t}dtB \end{aligned} \tag{5.15}$$

Macierz obserwatora stanu H pozwala na transformację wartości mierzonych do postaci wektora stanu. W celu jej obliczenia należy stworzyć odpowiednią formę macierzy, która pozwoli na spełnienie następującego równania:

$$z_n = Hx_n + v_n \tag{5.16}$$

gdzie z_n to wektor pomiarów w chwili n, H to macierz obserwatora, x_n to wektor stanu, natomiast v_n to niemierzalna wartość błędów pomiarowych.

Macierz kowariancji błędu pomiarowego R jest elementem uwzględniającym błąd pomiarowy w systemie. Została przedstawiona w równaniu 5.17:

$$R = \begin{bmatrix} \sigma^2_{z_1} & \sigma^2_{z_1,z_2} & \cdots & \sigma^2_{z_1,z_n} \\ \sigma^2_{z_2,z_1} & \sigma^2_{z_2} & \cdots & \sigma^2_{z_2,z_n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^2_{z_n,z_1} & \sigma^2_{z_n,z_2} & \cdots & \sigma^2_{z_n} \end{bmatrix} \tag{5.17}$$

gdzie σ to wariancja niepewności pomiarowej, z oznacza daną wartość mierzoną, natomiast n to liczba mierzonych wartości. Oznacza to, że na diagonali macierzy znajdują się wariancję danych wartości pomiarowych, a w pozostałych komórkach macierzy ich kowariancje. W celu obliczenia elementów tej macierzy należy określić wariancję poszczególnych wartości mierzonych oraz ich kowariancję poprzez stosowne wyliczenia lub symulacyjny dobór najlepszych wartości.

Macierz Q zwana macierzą kowariancji błędu procesu pozwala uwzględnić odchylenia obiektu rzeczywistego od modelu wynikające z różnych aspektów, które nie są możliwe do przewidzenia. Przykładowo, dla modelowania rakiet może być to wiatr, turbulencja lub, w przypadku samolotów, dodatkowy manewr wykonany przez pilota. Macierz Q została przedstawiona w równaniu 5.18:

$$Q = \begin{bmatrix} \sigma^2_{x_1} & \sigma^2_{x_1,x_2} & \cdots & \sigma^2_{x_1,x_n} \\ \sigma^2_{x_2,x_1} & \sigma^2_{x_2} & \cdots & \sigma^2_{x_2,x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^2_{x_n,x_1} & \sigma^2_{x_n,x_2} & \cdots & \sigma^2_{x_n} \end{bmatrix} \tag{5.18}$$

gdzie σ oznacza wariancję błędu, x_n oznacza daną zmienną wektora stanu, natomiast n to liczba zmiennych stanu. W celu określenia wartości macierzy należy obliczyć wariancje i kowariancje jej elementów. Wzory umożliwiające przeprowadzenie tych obliczeń są następujące:

$$V(x_n) = \sigma^2_{x_n} = E(x_n^2) - \mu^2_{x_n} \tag{5.19}$$

gdzie $V(x_n)$ oraz $\sigma^2_{x_n}$ to wariancja błędu procesu danej zmiennej stanu, $E(x_n^2)$ to wartość oczekiwana zmiennej losowej, w tej sytuacji zmiennej stanu, natomiast $\mu^2_{x_n}$ to średnia zmiennej losowej.

$$COV(x_n, x_m) = COV(x_m, x_n) = E(x_n x_m) - \mu_{x_n}\mu_{x_m} \tag{5.20}$$

gdzie $COV(x_n, x_m)$ oznacza kowariancję zmiennych stanu x_n i x_m, E to wartość oczekiwana zmiennej losowej, w tym przypadku pomnożonych przez siebie wartości zmiennych stanu, natomiast μ_{x_n} to średnia zmiennej losowej.

Macierz P zwana macierzą kowariancji estymacji stanu bieżącego. Jest określana wzorem 5.21:

$$P = \begin{bmatrix} p_x & p_{x\dot{x}} & \cdots & p_{xx^{(n)}} \\ p_{\dot{x}x} & p_{\dot{x}} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ p_{x^{(n)}x} & p_{x^{(n)}\dot{x}} & \cdots & p_{x^{(n)}x^{(n)}} \end{bmatrix} \tag{5.21}$$

gdzie p_x to wariancja estymacji pozycji zmiennej stanu x, natomiast $p_{x\dot{x}}$ to kowariancja zmiennych stanu x oraz $\dot{x}$. Ze względu na to, że w trakcie pracy filtra macierz P jest automatycznie dostrajana, najlepszą praktyką określenia wartości tej macierzy jest ogólna estymacja dokonana przez inżyniera, z założeniem że jeśli estymacja wektora zmiennych stanu jest trafna, należy jej przypisać większą wartość [1].

5.2.4 Opracowanie modelu [1]

Prace nad algorytmem rozpoczęto od przygotowania matematycznego modelu wysokości lotu rakiety pod postacią równań stanu. Z racji faktu, że w podręczniku *"Kalman Filter from the Ground Up„* Alexa Beckera [1] przedstawiono korespondujący z modelowaną rakietą proces wyprowadzenia modelu ciała poruszającego się ze stałym przyspieszeniem, rozdział ten został umieszczony w części teoretycznej pracy. Przyjęto założenie lotu w jednym wymiarze z uwagi na fakt, że algorytm nie wymaga estymacji toru lotu w przestrzeni trójwymiarowej. Na rysunku 5.1 przedstawiony został wykres zmiany wysokości lotu rakiety w przyjętym modelu, wygenerowany za pomocą symulatora OpenRocket opisanego w rozdziale 6.1.1.

Rysunek 5.1: Wykres zmiany wysokości lotu rakiety w przyjętym modelu.

Na wykresie przedstawione zostały również główne wydarzenia w trakcie lotu rakiety, min. zapłon silnika, wypalenie czy apogeum.

Następnie przyjęto wektor zmiennych stanu składający się z pozycji rakiety względem poziomu powierzchni ziemi w kierunku prostopadłym do niej i zwrocie w stronę przestrzeni powietrznej oraz prędkości względem ziemi. Z uwagi na jednowymiarowość modelu, do równań stanu zaadoptowano tylko wartości odnoszące się do osi natarcia rakiety. Ponadto, w wektorze nie zostało uwzględnione przyspieszenie rakiety ze względu na to, że zmienna ta jest mierzona i wykorzystana jako wejście do systemu. W równaniu 5.22 został przedstawiony model obiektu [1]:

$$\begin{aligned} \begin{bmatrix} \dot{h} \\ \dot{v} \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} h \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} a \\ h &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} h \\ v \end{bmatrix} \end{aligned} \tag{5.22}$$

gdzie $\dot{h}$ oznacza pochodną położenia (wysokości lotu), $\dot{v}$ oznacza pochodną prędkości, h oznacza wysokość lotu, v oznacza prędkość, natomiast a oznacza przyspieszenie.
Z powyższego równania można wyznaczyć macierze A i B będące elementami modelu obiektu w postaci równania stanu:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \; B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \; C = \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{5.23}$$

Jak wspomniano w rozdziale 5.2.3, aby określić macierze F i G wykorzystywane w filtrze Kalmana należy rozwiązać poniższe równanie różniczkowe:

$$x_{n+1} = Fx_n + Gu_n \tag{5.24}$$

W celu obliczenia macierzy F, macierz wykładnicza przedstawiona we wzorze 5.14 może zostać przedstawiona jako rozwinięcie wzoru Taylora:

$$e^{\boldsymbol{X}} = \sum_{k=0}^{\infty} \frac{1}{k!} \boldsymbol{X}^k \tag{5.25}$$

w związku z czym, macierz F może zostać obliczona w następujący sposób:

$$F = e^{A\Delta t} = I + A\Delta t + \frac{(A\Delta t)^2}{2!} + \frac{(A\Delta t)^3}{3!} \dots \tag{5.26}$$

gdzie:

$$A^2 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = 0$$

Z racji tego, że macierz A^2 jest równa zero, wzór 5.26 można skrócić do następującej postaci:

$$F = I + A\Delta t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \Delta t = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \tag{5.27}$$

a więc:

$$F = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$$

W celu obliczenia macierzy G ogólną formułę równania przedstawionego w rozdziale 5.2.3 można przedstawić w następującej postaci:

$$\int_0^{\Delta t} e^{At} dt = \Delta t \left(I + \frac{A\Delta t}{2!} + \frac{(A\Delta t)^2}{3!} + \dots \right) \tag{5.28}$$

gdzie:

$$A^2 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = 0$$

w związku z czym:

$$\int_0^{\Delta t} e^{At} dt = \Delta t \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \frac{\Delta t}{2} \right) = \begin{bmatrix} \Delta t & \frac{1}{2}\Delta t^2 \\ 0 & \Delta t \end{bmatrix}$$

$$G = \int_0^{\Delta t} e^{At} dt = \Delta t B = \begin{bmatrix} \Delta t & \frac{1}{2}\Delta t^2 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix}$$

A więc:

$$G = \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} \tag{5.29}$$

Co pozwala na przedstawienie równania ekstrapolacji stanu w następującej formie:

$$\begin{bmatrix} h_{n+1} \\ v_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} h_n \\ v_n \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} a \tag{5.30}$$

W celu obliczenia macierzy Q należy obliczyć wartości wariancji zmiennych stanu h i v oraz ich kowariancji:

$$V(h) = E(h^2) - \mu_x^2 = E\left(\left(\frac{1}{2}a\Delta t^2\right)^2\right) - \left(\frac{1}{2}\mu_a\Delta t^2\right)^2$$
$$= \frac{\Delta t^4}{4}(E(a^2) - \mu_a^2) = \frac{\Delta t^4}{4}\sigma_a^2$$

$$V(v) = E(x^2) - \mu_v^2 = E((a\Delta t)^2) - (\mu_a\Delta t)^2 = \Delta t^2(E(a^2) - \mu_a^2) = \Delta t^2\sigma_a^2$$

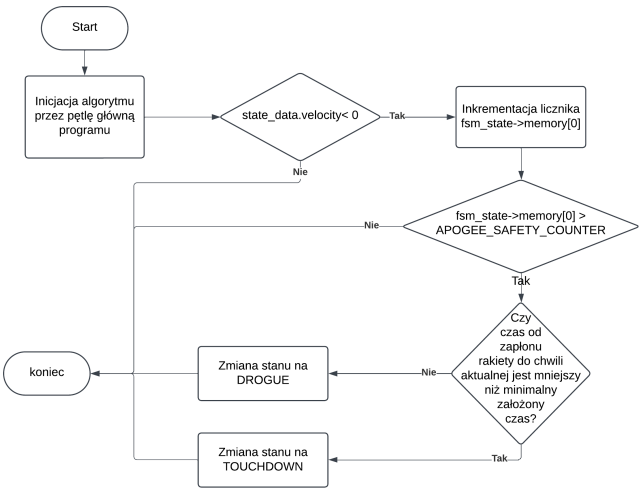
$$COV(h,v) = COV(v,h) = E(xv) - \mu_x\mu_v$$
$$= E\left(\frac{1}{2}a\Delta t^2 a\Delta t\right) - \left(\frac{1}{2}\mu_a\Delta t^2\mu_a\Delta t\right) = \frac{\Delta t^3}{2}(E(a^2) - \mu_a^2) = \frac{\Delta t^3}{2}\sigma_a^2$$

Co, po podsumowaniu, daje następującą macierz Q:

$$Q = \sigma_a^2\begin{bmatrix}\frac{\Delta t^4}{4} & \frac{\Delta t^3}{2}\\ \frac{\Delta t^3}{2} & \Delta t^2\end{bmatrix} \tag{5.31}$$

5.3 Metody wykrywania faz lotu na podstawie wykrycia apogeum [3]

Wykrycie apogeum lotu rakiety jest jednym z głównych zadań niniejszej pracy. Analiza kodu źródłowego komercyjnego komputera pokładowego CATS pozwala na sprawdzenie, w jaki sposób w praktyce wykorzystywane są tego typu algorytmy. Schemat blokowy algorytmu wygląda w sposób następujący [3]:



Rysunek 5.2: Schemat blokowy algorytmu wykrywania apogeum rakiety z komputera pokładowego CATS

gdzie `state_data.velocity` to zmienna systemu przechowująca aktualną wartość estymowanej prędkości, natomiast `fsm_state->memory[0]` to zmienna, w której przechowywany jest licznik pomocniczy do określenia apogeum.

Algorytm, po wywołaniu przez pętlę główną programu, sprawdza, czy prędkość rakiety jest mniejsza od zera. Jeśli tak, następuje inkrementacja licznika pomocniczego. Jeśli osiągnie on prędkość większą niż określona w zmiennej `APOGEE_SAFETY_COUNTER`, ustalonej na stałe jako 30 próbek, sprawdzany jest czas pomiędzy zapłonem silnika a chwilą aktualną. Jeśli czas jest mniejszy, oznacza to, że nastąpiła awaria i rakieta znajduje się na ziemi, następuje zmiana fazy na `TOUCHDOWN`.

Natomiast jeśli obliczony czas jest większy, apogeum zostaje wykryte i następuje zmiana stanu na TOUCHDOWN.

Poniżej została przedstawiona implementacja algorytmu w kodzie [3]:

```
static void check_coasting_phase(flight_fsm_t *fsm_state,
                                 estimation_output_t state_data)
{
  /* When velocity is below 0, coasting concludes */
  if (state_data.velocity < 0) {
    fsm_state->memory[0]++;
  }

  if (fsm_state->memory[0] > APOGEE_SAFETY_COUNTER) {
    /* If the duration between thrusting and apogee is smaller than defined*/
    /* go to touchdown */
    if ((osKernelGetTickCount() - fsm_state->thrust_trigger_time)
         < MIN_TICK_COUNTS_BETWEEN_THRUSTING_APOGEE) {
      change_state_to(TOUCHDOWN, EV_TOUCHDOWN, fsm_state);
    } else {
      change_state_to(DROGUE, EV_APOGEE, fsm_state);
    }
  }
}
```

gdzie:

- state_data to struktura przechowująca wartości zmiennych stanu, na przykład prędkości (state_data.velocity)
- fsm_state to struktura przechowująca dane na temat fazy lotu
- APOGEE_SAFETY_COUNTER to zmienna przechowująca liczbę określającą ilość próbek pozwalających na określenie zmiany stanu rakiety
- osKernelGetTickCount() to funkcja pobierająca aktualną wartość zegara systemu
- MIN_TICK_COUNTS_BETWEEN_THRUSTING_APOGEE to zmienna zawierająca liczbę określającą minimalny dopuszczalny czas między rozpoczęciem fazy wystrzału rakiety a apogeum
- change_state_to to funkcja pozwalająca na zarejestrowanie zmiany fazy lotu w strukturze fsm_state

Algorytm działa w sposób następujący: funkcja check_coasting_phase wyoływana jest w każdej iteracji głównej pętli programu podczas fazy wybiegu rakiety. Jeśli wartość prędkości estymowana za pomocą filtru Kalmana osiąga wartość mniejszą niż zero, rozpoczyna się zliczanie ilości próbek, które spełniają ten warunek, a kolejne wartości licznika zapisywane są w zmiennej fsm_state->memory[0]. Jeśli liczba spełniających ten warunek próbek przekroczy wartość określoną przez zmienną APOGEE_SAFETY_COUNTER, moment ten zostanie uznany za osiągnięcie apogeum lotu, o ile czas od startu rakiety do fazy apogeum nie jest mniejszy niż wartość określona w zmiennej MIN_TICK_COUNTS_BETWEEN_THRUSTING_APOGEE wynoszącej 1500 tyknięć zegara jądra systemu, co oznaczałoby, że nastąpiła nieprzewidziana sytuacja w locie lub błąd estymacji.

5.4 Pomiary ciśnienia przy przekroczeniu bariery dźwięku [7]

Z uwagi na fakt, że rakieta Hexa jest w stanie przekroczyć barierę dźwięku, niezbędne jest wzięcie pod uwagę fali uderzeniowej powstającej przy tego typu zjawisku. Fala ta jest cienką warstwą zaburzonego powietrza tworzącą się na powierzchni samolotu. W przypadku przejścia tej fali przez barometr, pomiary ciśnienia mogą zawierać wartości błędne - inne niż faktyczne ciśnienie atmosferyczne na danej wysokości, co spowoduje niepoprawne obliczenie wysokości oraz estymację prędkości, która, ze względu na nagłą, pozorną zmianę pozycji rakiety, osiągnie dużą wartość. Błędy te pojawiają się w trakcie fazy lotu z prędkością zbliżoną lub powyżej prędkości dźwięku, która dalej będzie nazywana transoniką. Z tego względu niezbędne jest uwzględnienie błędów, które mogą pojawić się w trakcie fazy transoniki.

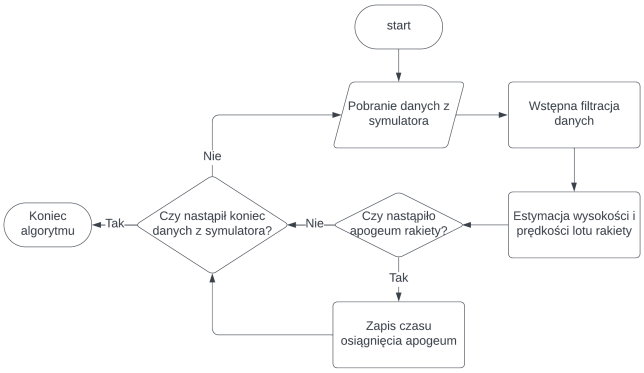
5.5 Symulator lotu OpenRocket [12]

OpenRocket to program umożliwiający tworzenie modelu symulacyjnego rakiety z uwzględnieniem takich czynników, jak gęstość materiałów, jakość wykonania powłoki zewnętrznej, człony rakiety a nawet trójwymiarowy model CAD jej elementów. Dzięki temu istnieje możliwość przeprowadzenia testów i symulowanych lotów pojazdu przed jego zbudowaniem w celu optymalizacji i ulepszenia komponentów. W trakcie tworzenia algorytmu wykorzystano dane wygenerowane z tego programu za pomocą modelu dostarczonego przez członków koła. Dane wyjściowe symulatora zostały przedstawione w rozdziale 6.1.1.

Rozdział 6

Proces opracowania algorytmu

Pierwszą częścią pracy wykonanej w ramach projektu było opracowanie algorytmu w języku programowania *Python*. Założenia algorytmu zostały przedstawione na rysunku 6.1. Założenia te zostały określone przez autora pracy wspólnie z członkami koła naukowego.



Rysunek 6.1: Schemat blokowy koncepcji algorytmu

gdzie wstępna filtracja danych oznacza filtrację sygnału w celu eliminacji błędów pomiarowych i uwzględnienie zwiększenia błędów pomiarowych pojawiających się w transonice opisanych w rozdziale 5.4.
Koncepcja algorytmu zakłada, że będzie on działał w pętli, na początku której za każdym razem będą pobierane dane z symulatora. Następnie, po wstępnej filtracji danych nastąpi estymacja wysokości i prędkości lotu rakiety, niezbędna do wykrycia apogeum. Dalej, jeśli zostanie ono wykryte, zostanie zapisany czas wystąpienia tego wydarzenia. Pętla kończy się sprawdzeniem, czy nastąpił koniec danych z symulatora. Jeśli tak, praca algorytmu kończy się. Jeśli nie, odbywa się kolejna iteracja pętli programu.

Podjęto decyzję, aby algorytm oprzeć przede wszystkim na filtrze Kalmana, ponieważ spełnia on zarówno funkcję filtru, pozwalającego na pozbycie się z programu błędów pomiarowych oraz ponieważ w bardzo dobry sposób pozwala na estymację niedostępnych pomiarowo zmiennych stanu [1]. Ponadto, możliwość zmiany niektórych nastaw regulatora, np. wariancji pomiarowej, umożliwia uodpornienie algorytmu na zmienne poziomy zakłóceń błędów pomiarowych.

6.1 Implementacja LKF

Poniżej została przedstawiona implementacja algorytmu liniowego filtru Kalmana. Jest to klasa `KalmanFilter` pozwalająca na inicjalizację macierzy filtru oraz przeprowadzenie niezbędnych obliczeń.

```
import numpy as np

class KalmanFilter:
    def __init__(self, F, G, H, Q, P, x):
        self.F = F  # State transition matrix
        self.G = G  # Control input matrix
        self.H = H  # Observation matrix
        self.Q = Q  # Process noise covariance
        self.P = P  # Estimate error covariance
        self.x = x  # State estimate

    def predict(self, u):
        # Predict the state and estimate covariance
        # x = F*X + G*u
        self.x = self.F @ self.x + self.G * u

        # P_pred = F*P*A_T + Q
        self.P = self.F @ self.P @ self.F.T + self.Q

    def update(self, z, R):
        self.R = R
        # Kalman gain
        # K = P*H.T * inv(H * P * H.T + R)
        K = self.P @ self.H.T / (self.H @ self.P @ self.H.T + self.R)

        # Update the state estimate
        # x_curr = x*K*(z-H*x)
        self.x = self.x + K * (z - self.H @ self.x)

        # Update the estimate covariance
        # P_curr = (I-K*H) * P (I-KH).T + K*R*K.T
        self.P = (np.eye(self.P.shape[0]) - K * self.H) @ self.P @ (np.eye(self.P.shape[0]) - K

        # P matrix trace
        P_trace = np.trace(self.P)

        return self.x[0][0], self.x[1][0], P_trace
```

Podczas inicjacji obiektu tej klasy przekazywane i zapisywane są stałe wartości macierzy F, G, H oraz Q, a także początkowe wartości macierzy P, R oraz wektora zmiennych stanu x. Klasa definiuje

również trzy funkcje wewnętrzne:

- predict(self, u) - funkcja przeprowadzająca obliczenia równań predykcji. Jako wartość wejściową tej funkcji podaje się zmienną u wejścia systemu, którą w tym przypadku jest przyspieszenie rakiety mierzone akcelerometrem.
- update(self, z, R) - funkcja przeprowadzająca obliczenia równań aktualizacji. Jako wartość wejściową funkcji podaje się zmienną z, do której przypisana jest akutalna wartość pomiaru wysokości oraz zmienną R, do której przypisana została wartość macierzy kowariancji błędu pomiarowego.

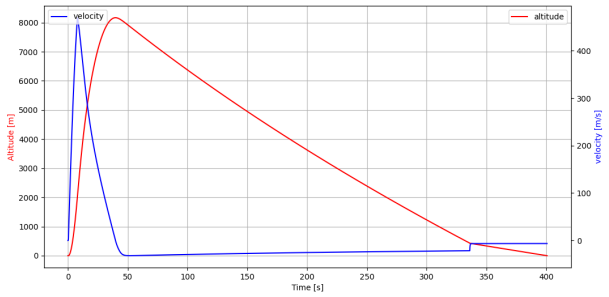
W celu efektywnego przeprowadzenia obliczeń zastosowano operator mnożenia macierzy `@` zaproponowany w dokumencie projektu oprogramowania PEP 465 i wprowadzony do języka Python w wersji 3.5. Ponadto, klasa używa biblioteki `numpy` aby zaimplementować macierz jednostkową `np.eye()` oraz funkcję obliczenia śladu macierzy `np.trace()`.

6.1.1 Dane z symulacji

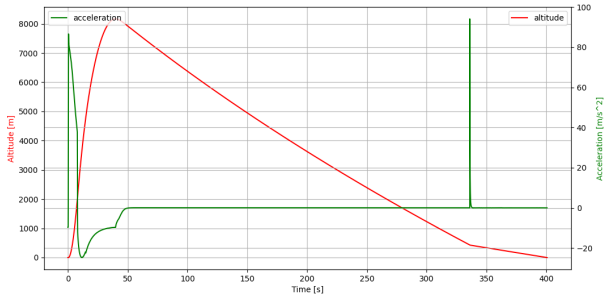
Przeprowadzenie testów algorytmu wymagało danych lotu rakiety, które umożliwią sprawdzenie poprawności koncepcji oraz obliczeń. W tym celu wykorzystano symulator **OpenRocket** z dostarczonymi przez członków koła modelami rakiety oraz przygotowanymi testami. Program ten został opisany w rozdziale 5.5. Z symulacji zostały pobrane następujące dane:

- Próbki czasu symulacji
- Wysokość lotu rakiety - kolor czerwony
- Prędkość chwilowa rakiety - kolor niebieski
- Przyspieszenie chwilowe rakiety - kolor zielony
- Ciśnienie atmosferyczne w otoczeniu rakiety - kolor fioletowy

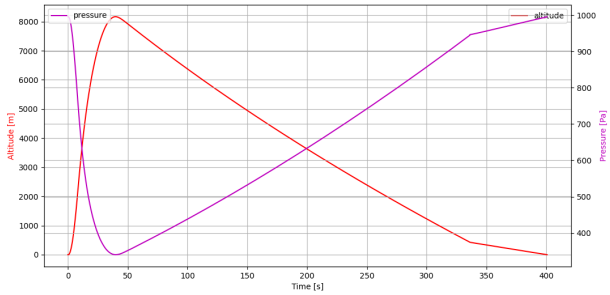
Dane zostały przedstawione na rysunku 6.2.



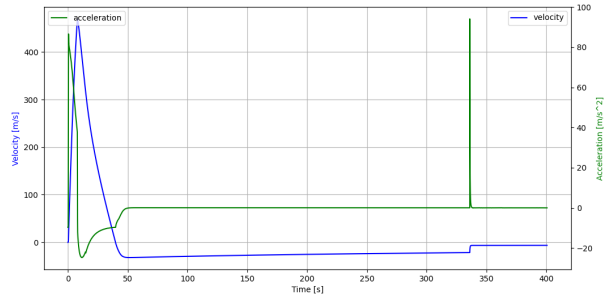
(A) Wykres przedstawiający wysokość lotu oraz prędkość rakiety



(B) Wykres przedstawiający wysokość lotu oraz przyspieszenie rakiety



(C) Wykres przedstawiający wysokość lotu rakiety oraz ciśnienie atmosferyczne



(D) Wykres przedstawiający prędkość oraz przyspieszenie rakiety

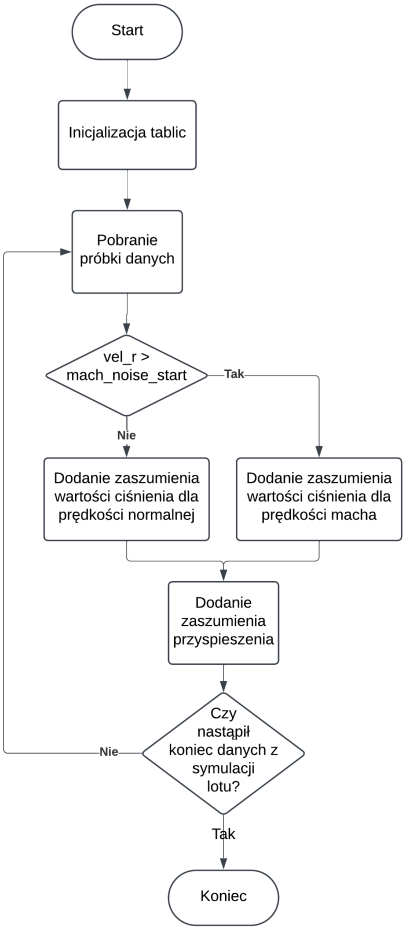
Rysunek 6.2: Wykresy prezentujące dane lotu z symulacji

Należy zwrócić uwagę na pewne aspekty ukazane na wykresach. W momencie osiągnięcia apogeum zarówno prędkość chwilowa jak i przyspieszenie osiągają wartość równą zero. Ponadto, wykres zmiany ciśnienia atmosferycznego ma kształt zbliżony do wykresu zmiany wysokości lotu w horyzontalnie lustrzanym odbiciu. Dodatkowo, szczytowa wartość przyspieszenia osiągana jest bardzo blisko momentu startu by następnie osiągnąć wartość ujemną. Moment, gdy wskazanie przyspie-

szenia wynosi zero, jest momentem wypalenia silnika i rozpoczyna fazę wybiegu rakiety. Ostatnim aspektem jest prędkość, która w chwili bliskiej momentu startu przekracza prędkość dźwięku. Na wykresach oznaczonych literami A oraz B, na wykreślonym przebiegu przyspieszenia widoczne są nagłe, wysokie zmiany wartości. Wynikają one z tego, że w tym momencie dokonywane jest włączenie dodatkowych spadochronów chroniących rakietę przed samym momentem upadku. Nagły, dodatkowy opór skutkuje zmniejszeniem prędkości chwilowej rakiety, która poddana jest silnemu, nagłemu spowolnieniu.

6.1.2 Algorytm dodania błędu pomiarowego [5]

Tak jak wspomniano w rozdziale 5.4, w przedziale czasu, gdy prędkość rakiety jest zbliżona lub wyższa od prędkości jednego macha, wskazania barometru mogą przedstawiać niepoprawne informacje. Niestety, symulator wykorzystany do uzyskania danych lotu rakiety nie uwzględnia takiego zjawiska. Z tego względu opracowano algorytm dodający odpowiednie wartości błędu pomiarowego. Na rysunku 6.3 umieszczony został schemat blokowy algorytmu zaszumiania danych z symulacji [5].



Rysunek 6.3: Schemat blokowy algorytmu zaszumiania danych z symulacji OpenRocket

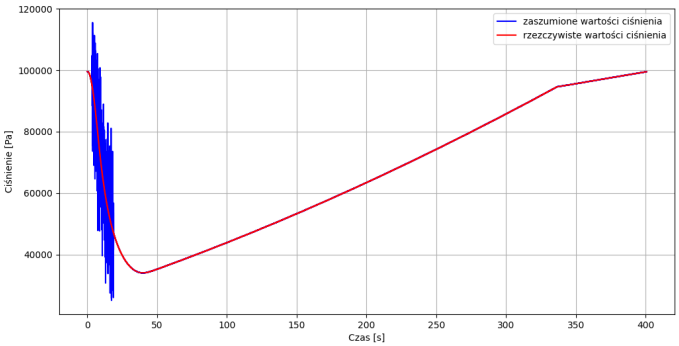
Algorytm ten pobiera dane uzyskane z symulatora OpenRocket, a następnie dodaje do nich wartości zaszumienia odczytów ciśnienia właściwe dla danej prędkości lotu. Następnie dodawane jest zaszumienie wartości przyspieszenia rakiety. Dalej sprawdzane jest, czy nastąpił koniec danych z symulacji lotu. Jeśli tak, algorytm kończy swoją pracę, jeśli natomiast nie, kontynuowane jest

działanie pętli.
Poniżej została przedstawiona implementacja algorytmu w języku Python:

```
def mach_noise_adder_gauss(original_data_csv, noise_presets, Tp, mach_noise_start, EventFlags):
    if Tp > 0.02:
        return ValueError("Tp must be less than or equal 0.02")
    timestamp = 0
    t_real    = []
    alt_real  = []
    vel_real  = []
    acc_real  = []
    acc_noise = []
    pre_real  = []
    pre_noise = []

    with open(original_data_csv, "r") as file:
        reader = csv.reader(file, delimiter=";")
        for row in reader:
            sim_time = float(row[0])
            alt_r    = float(row[1])
            vel_r    = float(row[2])
            acc_r    = float(row[3])
            pre_r    = float(row[4]) * 100 # mbar to Pa

            while timestamp < sim_time:
                # add noise
                if (vel_r > mach_noise_start):
                    pre_n = random.gauss(pre_r, noise_presets['press_mach_sig'])
                    if not EventFlags.mach_start_flag:
                        EventFlags.mach_start_flag = True
                        EventFlags.mach_start_time = timestamp
                else:
                    pre_n = random.gauss(pre_r, noise_presets['press_normal_sig'])
                    if EventFlags.mach_start_flag and not EventFlags.mach_end_flag:
                        EventFlags.mach_end_flag = True
                        EventFlags.mach_end_time = timestamp

                acc_n = random.gauss(acc_r, noise_presets['acc_sig'])

                # append data
                t_real.append(timestamp)
                alt_real.append(alt_r)
                vel_real.append(vel_r)
                acc_real.append(acc_r)
                acc_noise.append(acc_n)
                pre_real.append(pre_r)
                pre_noise.append(pre_n)
```

```
            timestamp += Tp

    return t_real, alt_real, vel_real, acc_real, acc_noise,
    pre_real, pre_noise, EventFlags
```

Funkcja mach_noise_adder_gauss(original_data_csv, noise_presets, Tp, mach_noise_start, EventFlags) jako wejście przyjmuje dane symulacji w postaci zapisu CSV, ustawienia początkowe wartości błędu, czas próbkowania, prędkość powyżej której rozpocznie się zaszumienie danych oraz klasę EventFlags zawierającą informacje o czasie rozpoczęcia i zakończenia wprowadzania wartości z szumem.
Wartość `noise_presets['acc_sig']` została określona na podstawie karty katalogowej czujnika BMI088 [11] i jest równa $2.5\frac{m}{s^2}$. W ramach działania, algorytm na początku sprawdza poprawną wartość próbkowania danych wprowadzoną do algorytmu głównego, następnie pobiera oryginalne dane lotu i do wartości próbek zasymulowanego pomiaru ciśnienia dodaje szum o rozkładzie Gaussa z odpowiednim odchyleniem standardowym, przekazywanym za pomocą zmiennej `noise_presets['press_normal_sig']` równej 20.5 Pa, określonej z błędu najgorszego przypadku z noty katalogowej czujnika [6] lub `noise_presets['press_mach_sig']` równej 10000 Pa, określonej tak, aby pomiary ciśnienia mogły wskazywać wartości zbliżone do zera oraz były nieproporcjonalnie wysokie do normalnych, niezaszumionych pomiarów. Na rysunku 6.4 został przedstawiony wykres zawierający zaszumione wartości pomiarów ciśnienia oraz przebieg rzeczywistych wskazań barometru.



Rysunek 6.4: Wykres rzeczywistych oraz zaszumionywch wartości ciśnienia

Na rysunku 6.4 można zauważyć, że na wykresie widoczne są głównie próbki zakłóconego ciśnienia właściwe dla lotu w fazie transoniki. Wynika to z tego, że zastosowana w pozostałych fazach lotu wartość zaszumienia σ_{normal} jest wartością zbyt małą, aby była widoczna dla danego zakresu wartości, w jakim wyskalowana jest oś rzędnych. Nie zmienia to jednak faktu, że błędy pomiarowe są obecne w trakcie lotu z prędkością z prędkością poniżej jednego macha i niefiltrowane wpływają negatywnie na obliczenia wysokości.

6.1.3 Obliczanie wysokości lotu rakiety z wartości ciśnienia

Filtr Kalmana, jako wartości mierzone, przyjmuje wysokość na jakiej znajduje się rakieta. Ponieważ wartość ta nie jest bezpośrednio dostępna, uzyskiwana jest poprzez obliczenia z zastosowaniem wskazań barometru. Poniżej przedstawiona została funkcja `pressure_preprocessing`, w której obliczenia te zostały zaimplementowane.

```
L = -0.0065
T0 = 15

def pressure_preprocessing(p_pa, p0_pa, alt0_m):
    alt = ( (p_pa/p0_pa)**(1/5.257) - 1 ) * (T0+273.15)/L - alt0_m

    return alt
```

Funkcja ta realizuje równanie o numerze 6.1 [4][13].

$$h_n = \left(\left(\frac{p_n}{p_0} \right)^{\frac{1}{k}} - 1 \right) \times \frac{(T_0 + 273,15)}{L} - h_0 \tag{6.1}$$

gdzie h_n to aktualna wartość wysokości lotu rakiety, p_n to aktualna wartość ciśnienia, p_0 to wartość ciśnienia przy powierzchni ziemi, k to współczynnik wyliczony z równania $\frac{-R \times L_b}{g_0 \times M}$ wiążącego uniwersalną stałą gazową, standardowy współczynnik spadku temperatury, przyśpieszenie grawitacyjne i masę molową ziemi, równy $k = 5,257$, L to standardowy współczynnik spadku temperatury, T_0 to standardowa temperatura na poziomie morza, natomiast h_0 to wysokość, z jakiej rakieta startuje. Funkcja jako dane wejściowe przyjmuje aktualną wartość ciśnienia, wartość ciśnienia przy powierzchni ziemi oraz wysokość początkową. Przyjmuje się następujące wartości stałe: $L = -0,0065$, $T_0 = 15°\mathrm{C}$

6.1.4 Inicjalizacja filtru Kalmana

Aby dokonać poprawnej inicjalizacji filtru Kalmana, należy wprowadzić odpowiednie wartości początkowe. Poniżej zostały przedstawione zainicjowane wartości macierzy filtru:

$$F = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$$

$$G = \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$Q = \sigma_a^2 \begin{bmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & \Delta t^2 \end{bmatrix}$$

gdzie wartość $\Delta t = 0.02$ s jest czasem iteracji pętli w projekcie, $\sigma_a^2 = 2.5^2 \ \frac{m}{s^2} = 6.25 \ \frac{m^2}{s^4}$ to wariancja błędu pomiaru z akcelerometru.
Ponadto, należy również podać początkowe wartości macierzy kowariancji błędu pomiarowego P oraz wektora zmiennych stanu x:

$$P_{init} = \begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}$$

$$x_{init} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Zgodnie z tym, co zostało opisane w rozdziale 5.2.3, wartość macierzy P została dobrana w sposób umiarkowany ze względu na dużą pewność trafności określenia wartości początkowej wektora zmiennych stanu x, ponieważ tor lotu rakiety wykreślany jest od pozycji startowej, która określana jest jako zerowa.

6.1.5 Dobór niepewności pomiarowej R

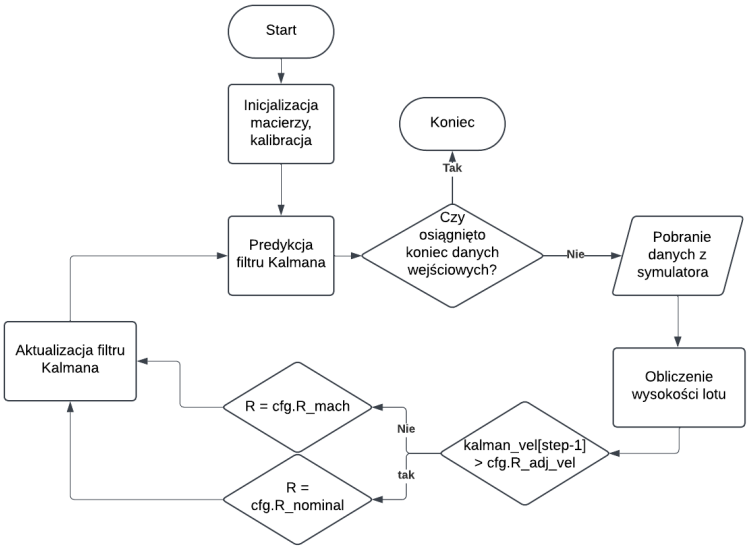
Ze względu na opisane w rozdziale 5.4 wysokie błędy pomiaru ciśnienia przy przekraczaniu bariery dźwięku, niezbędne uwzględnienie tego zjawiska w algorytmie. W tym celu dokonywana jest zmiana wartości macierzy R. Lot podzielono na dwie fazy, fazę normalną, oraz fazę lotu z prędkością blisko prędkości dźwięku. Zmiana fazy lotu skutkuje zmianą wartości macierzy R. Zmiana dokonywana jest na podstawie estymowanej przez filtr Kalmana prędkości. Jeśli zostanie przekroczona, zmieniana jest wartość. Proces ten jest wykonywany przez następujący algorytm:

```
if kalman_vel[step-1] > cfg.R_adj_vel:
        alt_k, vel_k, P_t = kalman.update( alt_meas[step], cfg.R_mach )
    else:
        alt_k, vel_k, P_t = kalman.update( alt_meas[step], cfg.R_nominal )
```

gdzie `kalman_vel[step-1]` to wartość prędkości estymowanej w poprzedniej iteracji algorytmu, `cfg.R_adj_vel` to parametr progowej wartości prędkości, której przekroczenie skutkuje zmianą parametru R, określonej na połowę prędkości dźwięku czyli 170 $\frac{m}{s}$. wartość ta została dobrana w taki sposób, aby zapewnić pewien margines błędu. Funkcja `kalman.update` realizuje obliczenia równań aktualizacji filtru Kalmana, natomiast `cfg.R_nominal = 100` i `cfg.R_mach = 10000000` to wartości niepewności pomiarowej R. Wartość `cfg.R_mach = 10000000` jest na tyle duża, że przez całą fazę lotu z prędkością zbliżoną lub powyżej jednego macha algorytm wykorzystuje przede wszystkim obliczenia wynikające z modelu, natomiast pomiary ciśnienia wykorzystywane są z bardzo niską ufnością, przez co wzmocnienie Kalmana jest bliskie zeru.

6.1.6 Schemat blokowy algorytmu liniowego

Na rysunku 6.5 przedstawiony został schemat blokowy programu `apogee_detection-LKF`



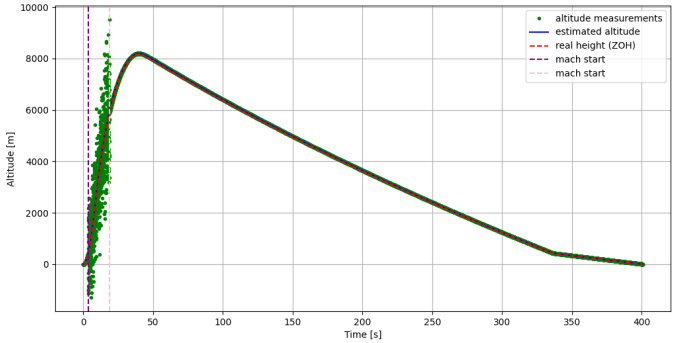
Rysunek 6.5: Schemat blokowy programu zawierającego zaimplementowany algorytm z wykorzystaniem liniowego filtru Kalmana

Po rozpoczęciu działania algorytmu następuje inicjalizacja wartości macierzy oraz zmiennych wewnętrznych programu. Następnie dokonywane są obliczenia predykcji filtru Kalmana. W dalszej kolejności sprawdzane jest, czy są dostępne nowe dane z symulacji. Jeśli nie, program się kończy, jeśli tak, pobierana jest następna próbka danych. W dalszej kolejności wartości ciśnienia przeliczane

są na wysokość. Następnie, na podstawie estymacji prędkości w poprzedniej iteracji podejmowana jest decyzja o tym, jaka wartość ma zostać przypisana do zmiennej R. Potem dokonywane są obliczenia aktualizacji filtru Kalmana. W tym miejscu następuje powtórzenie pętli i obliczenie równań predykcji.

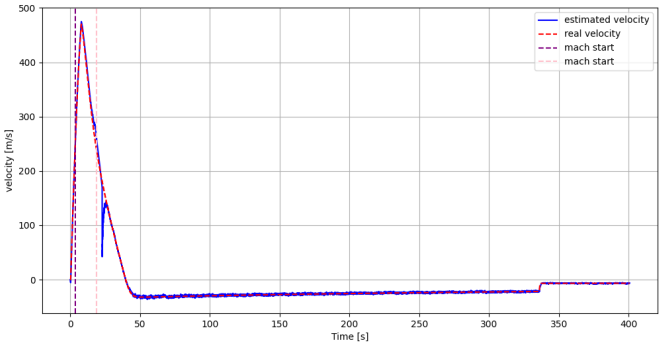
6.1.7 Wyniki, analiza i ewaluacja LKF

W tym akapicie zostały zaprezentowane wykresy prezentujące wyniki programu `apogee_detection-LKF` zawierającego implementację liniowego filtru Kalmana z symulacją błędu pomiarowego w transonice z rozkładem Gaussa. Na wszystkich wykresach przerywaną linią w kolorze fioletowym przedstawiono moment, w którym algorytm symulujący zakłócenia pomiarowe zmienia wartość szumu na reprezentującą możliwe wartości zaszumienia w fazie transoniki, natomiast przerywana linia w kolorze różowym reprezentuje koniec tej fazy.



Rysunek 6.6: Wykres porównujący estymowane, rzeczywiste i zmierzone wartości wysokości lotu rakiety

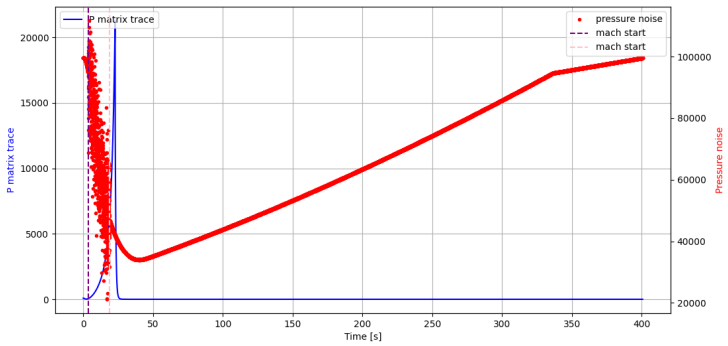
Wykres przedstawiony na rysunku 6.6 prezentuje estymację wysokości lotu rakiety oraz jej rzeczywisty przebieg. Wykresy te pokrywają się tworząc jedną linię, co oznacza bardzo dużą dokładność. Ponadto, można na nim zaobserwować zmierzone wartości ciśnienia oraz obszar cechujący się wysokim błędem pomiarowym.



Rysunek 6.7: Wykres porównujący estymowane i rzeczywiste wartości prędkości rakiety

Na wykresie przedstawionym na rysunku 6.7 zaprezentowana została estymacja prędkości rakiety. Czerwoną, przerywaną linią wykreślona jest rzeczywista prędkość, podczas gdy estymacja zaznaczona jest kolorem niebieskim. Wyniki te prezentują wysoką dokładność, poza jednym odchyleniem,

znajdującym się tuż za momentem wyjścia z fazy lotu z prędkością zbliżoną do prędkości dźwięku. Efekt ten powiązany jest z resetem macierzy P, opisanym przy ostatnim wykresie.



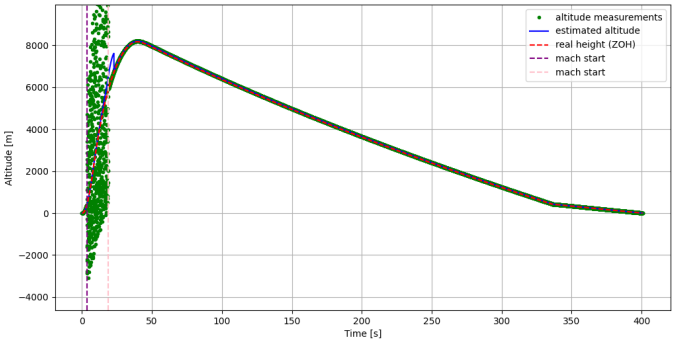
Rysunek 6.8: Wykres zestawiający wartości zmierzonego ciśnienia oraz wykreślony przebieg zmiany śladu macierzy P

Na wykresie 6.8 umieszczone zostało porównanie zaszumionych pomiarów ciśnienia oraz przebiegu zmiany śladu macierzy P. Widoczny jest wykładniczy wzrost obliczonego śladu macierzy od wartości bliskiej zeru do bardzo wysokiej. Następnie następuje gwałtowna zmiana i spadek przebiegu. Dzieje się to w momencie, gdy w algorytmie zmieniana jest wartość zmiennej R z odpowiadającej fazie lotu z prędkością zbliżoną do prędkości dźwięku na wartość nominalną. Wartość dla prędkości macha jest większa od zwykłej wartości sto tysięcy razy, następuje więc nagła zmiana śladu macierzy P. Razem ze zmianą wartości śladu macierzy następuje zwiększenie ufności pomiarów ciśnienia i dokonuje się aktualizacja wysokości lotu rakiety. Jest to nagła zmiana pozycji i skutkuje wysoką estymacją prędkości, co powoduje widoczne na wykresie 6.7 odchylenie wartości prędkości. W celu ewaluacji jakości filtra obliczono również wskaźnik RMSE na podstawie równania 6.2 [10] :

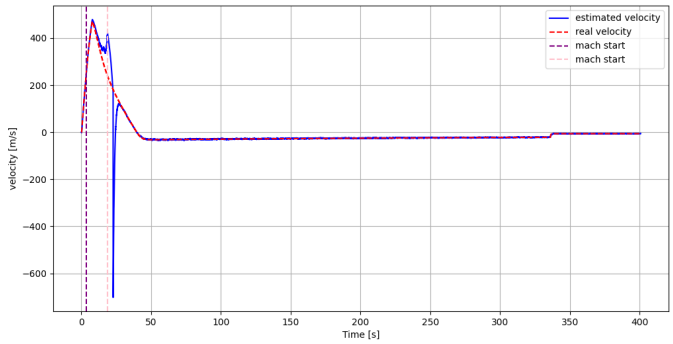
$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h_i - \hat{h}_i)^2} \tag{6.2}$$

Średnia wartość RMSE dla LKF wynosi 27.5.

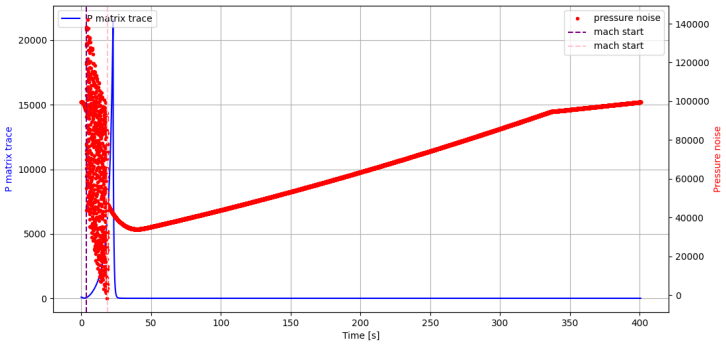
Wykresy przedstawione powyżej zostały wygenerowane z błędem pomiarowym w fazie transoniki zaszumionym losowymi wartościami o rozkładzie Gaussa. Jednak w trakcie testowania i ewaluacji algorytmu stwierdzono, że zdecydowanie bardziej trafnym założeniem jest uznanie jednorodnego rozkładu błędu. Poniżej przedstawiono wykresy dla LKF w tej sytuacji:



Rysunek 6.9: Wykres porównujący estymowane, rzeczywiste i zmierzone wartości wysokości lotu rakiety



Rysunek 6.10: Wykres porównujący estymowane i rzeczywiste wartości prędkości rakiety



Rysunek 6.11: Wykres zestawiający wartości zmierzonego ciśnienia oraz wykreślony przebieg zmiany śladu macierzy P

Jak widać na wykresie 6.9, estymacja wysokości w fazie lotu z wysokimi błędami pomiarowymi cechuje się odchyleniem od rzeczywistego przebiegu wysokości lotu. W po zakończeniu fazy transoniki następuje aktualizacja i powrót do poprawnego toru lotu. Średnia wartość wskaźnika RMSE obliczona dla tego algorytmu wynosi 85.2. Ponadto, wykres przebiegu prędkości przedstawiony na grafice 6.10 charakteryzuje się bardzo dużym spadkiem wartości, pojawiającym się w momencie wspomnianej aktualizacji położenia. Ze względu na to, że prędkość ta spada poniżej wartości zera, apogeum lotu mogłoby zostać błędnie wykryte. Z tego względu niezbędna jest zmiana zastosowanego algorytmu, który pozwoli poprawienie wyników.

6.2 Implementacja EKF

6.2.1 Linearyzacja funkcji obliczającej zmienną stanu na podstawie pomiaru

Ze względu na problemy opisane w rozdziale 6.1.7, zdecydowano się na zastosowanie algorytmu rozszerzonego filtru Kalmana. Zgodnie z rozdziałem 5.2.2, w którym został opisany, przyjęto, że w nowym algorytmie należy dokonać linearyzacji macierzy obserwatora H i wpisania w nią równania obliczania zmiennych wektora stanu z pomiarów ciśnienia. W tym celu z równania 6.1 należy wyprowadzić równanie na aktualną wartość ciśnienia p_n. Zostało ono przedstawione w równaniu 6.3.

$$p_n = p_0 \sqrt[k]{\frac{(h_n + h_0)L}{T_0 + 273,15} + 1} = h(x) \tag{6.3}$$

$$h(x_n) = z_n$$

gdzie h_n to aktualna wartość wysokości lotu rakiety, p_n to aktualna wartość ciśnienia, p_0 to wartość ciśnienia przy powierzchni ziemi, k to współczynnik wyliczony z równania $\frac{-R \times L_b}{g_0 \times M}$ wiążącego uniwersalną stałą gazową, standardowy współczynnik spadku temperatury, przyśpieszenie grawitacyjne i masę molową ziemi, równy $k = 5,257$, L to standardowy współczynnik spadku temperatury, T_0 to standardowa temperatura na poziomie morza, natomiast h_0 to wysokość, z jakiej rakieta startuje. Funkcja jako dane wejściowe przyjmuje aktualną wartość ciśnienia, wartość ciśnienia przy powierzchni ziemi oraz wysokość początkową. Przyjmuje się następujące wartości stałe: $L = -0,0065$, $T_0 = 15°\text{C}$. Funkcja $h(x_n)$ jest funkcją zastępującą macierz H. Następnie należy obliczyć Jakobian wyprowadzonej funkcji, zgodnie ze wzorem 6.4 [1].

$$\frac{\partial h(x)}{\partial x} = \begin{bmatrix} \frac{\partial h(x)}{\partial h} & \frac{\partial h(x)}{\partial \dot{h}} \end{bmatrix} \tag{6.4}$$

Obliczony Jakobian funkcji $h(x_n)$ został przedstawiony na równaniu 6.5.

$$\frac{\partial h(x)}{\partial x} = \begin{bmatrix} \frac{5,257 * L * P_0}{T_0 + 273,15} \left(\frac{L(h_0+h)+T_0+273,15}{T_0+273,15} \right)^{4,257} & 0 \end{bmatrix} \tag{6.5}$$

W związku z zastosowaniem linearyzacji, zrezygnowano z funkcji `pressure_preprocessing`. Równanie 6.4 zostało zaimplementowane w funkcji predykcji w klasie `KalmanFilter_class`, która po aktualizacji wygląda w następujący sposób:

```
def predict(self, u):
    # Predict the state and estimate covariance
    # x = F*X + G*u
    self.x = self.F @ self.x + self.G * u

    # P_pred = F*P*A_T + Q
    self.P = self.F @ self.P @ self.F.T + self.Q

def update(self, z, R):
    # R update
    self.R = R

    # linearize the observation model
    h_x = self.h_x(self.x[0][0])
    dh_dx = self.dh_dx(self.x[0][0])

    # Kalman gain
    # K = P*H.T * inv(H * P * H.T + R)
    K = self.P @ dh_dx.T / (dh_dx @ self.P @ dh_dx.T + self.R)

    # Update the state estimate
    # x_curr = x*K*(z-H*x)
    self.x = self.x + K * (z - h_x)

    # Update the estimate covariance
    # P_curr = (I-K*H) * P (I-KH).T + K*R*K.T
    self.P = (np.eye(self.P.shape[0]) - K * dh_dx) @ self.P @
```

```
        @ (np.eye(self.P.shape[0]) - K * dh_dx).T + (K*self.R@K.T)

        # P matrix trace
        P_trace = np.trace(self.P)

        return self.x[0][0], self.x[1][0], P_trace

    ''' UKF additional functions '''
    def h_x(self, alt_pred):
        h_x = self.p0 * (1+(self.L/self.T_K)*(alt_pred-self.alt0))**5.2558

        return h_x

    def dh_dx(self, alt_pred):
        dh_dhn = ( 5.257*self.L*self.p0* ( np.abs(((self.L*(alt_pred+self.alt0)) \
        / (self.T_K))+1 ))**4.257 ) / (self.T_K)

        dh_dx = np.array([dh_dhn, 0.0])

        return dh_dx
```

Zmianami wprowadzonymi w funkcji są zmienne `h_x` oraz `dh_dx` zawierające wartości obliczone za pomocą funkcji `h_x`(self, alt_pred) oraz `dh_dx`(self, alt_pred) realizujące równania 6.3 i 6.5. Model obiektu pozostaje bez zmian względem liniowego filtru Kalmana.

6.2.2 Ulepszona funkcja zaszumiająca

Z powodu zastosowania różnego rodzaju podejść do symulowania błędów pomiarowych ciśnienia została stworzona nowa funkcja mająca na celu jak najdokładniejsze odwzorowanie rzeczywistych warunków. Została ona przedstawiona poniżej:

```
def precise_noise_adder(original_data_csv, noise_presets, Tp,
mach_noise_start, RealEvents, start_mach_phase):
if Tp > 0.02:
    return ValueError("Tp must be less than or equal 0.02")
timestamp   = 0
t_real      = []
alt_real    = []
vel_real    = []
acc_real    = []
acc_noise   = []
pre_real    = []
pre_noise   = []

with open(original_data_csv, "r") as file:
    reader = csv.reader(file, delimiter=";")
    for row in reader:
        sim_time = float(row[0])
```

```
alt_r    = float(row[1])
vel_r    = float(row[2])
acc_r    = float(row[3])
pre_r    = float(row[4])

hf_range_n = noise_presets['press_mach_hf_range']
normal_sig = noise_presets['press_normal_sig']

while timestamp < sim_time:
    # add noise
    if (vel_r > 0.9*340):
        pre_n = random.uniform(pre_r-hf_range_n, pre_r+hf_range_n)
    elif (vel_r > mach_noise_start):
        noise_range = normal_sig + (hf_range_n - normal_sig) *
        * ((vel_r - mach_noise_start) / (0.9 * 340 - mach_noise_start))
        pre_n = random.uniform(pre_r-noise_range, pre_r+noise_range)
        # add flag informing about mach start
        if not RealEvents.real_mach_n_start_flag:
            RealEvents.real_mach_n_start_flag = True
            RealEvents.real_mach_n_start_time = timestamp
    else:
        pre_n = random.gauss(pre_r, normal_sig)
        if RealEvents.real_mach_n_start_flag
        and not RealEvents.real_mach_n_end_flag:
            RealEvents.real_mach_n_end_flag = True
            RealEvents.real_mach_n_end_time = timestamp

    acc_n = random.gauss(acc_r, noise_presets['acc_sig'])
    if vel_r > start_mach_phase
    and not RealEvents.real_mach_phase_start_flag:
        RealEvents.real_mach_phase_start_flag = True
        RealEvents.real_mach_phase_start_time = timestamp
    elif vel_r < start_mach_phase
    and not RealEvents.real_mach_phase_end_flag
    and RealEvents.real_mach_phase_start_flag:
        RealEvents.real_mach_phase_end_flag = True
        RealEvents.real_mach_phase_end_time = timestamp

    elif vel_r <= 0
    and not RealEvents.real_apogee_flag
    and RealEvents.real_mach_phase_end_flag:
        RealEvents.real_apogee_flag = True
        RealEvents.real_apogee_time = timestamp

    # append data
    t_real.append(timestamp)
    alt_real.append(alt_r)
```

```
            vel_real.append(vel_r)
            acc_real.append(acc_r)
            acc_noise.append(acc_n)
            pre_real.append(pre_r)
            pre_noise.append(pre_n)

            timestamp += Tp

    return t_real, alt_real, vel_real, acc_real, acc_noise, pre_real, pre_noise
```

Funkcja ta, do danych wejściowych w trakcie lotu z prędkością poniżej jednego macha, dodaje symulowany błąd pomiarowy o rozkładzie Gaussa oraz błąd o rozkładzie jednorodnym w fazie transoniki. Prędkość progowa, przy której rozpoczyna się większe zaszumianie danych została ustawiona na 80% prędkości dźwięku. Od przekroczenia progu prędkości do momentu osiągnięcia 90% prędkości macha zakres zaszumienia wzrasta wprost proporcjonalnie do poziomu zadanego w ustawieniach programu, określonego zmienną `noise_presets['press_mach_hf_range']`.
Ponadto, w funkcji dodawane są flagi określające rzeczywisty czas rozpoczęcia zaszumiania danych z błędem dla lotu w fazie transoniki, określone zmiennymi `RealEvents.real_mach_n_start_time` oraz `RealEvents.real_mach_n_end_time`, a także flagi określające rzeczywiste rozpoczęcie fazy lotu z prędkością zbliżoną do prędkości dźwięku.

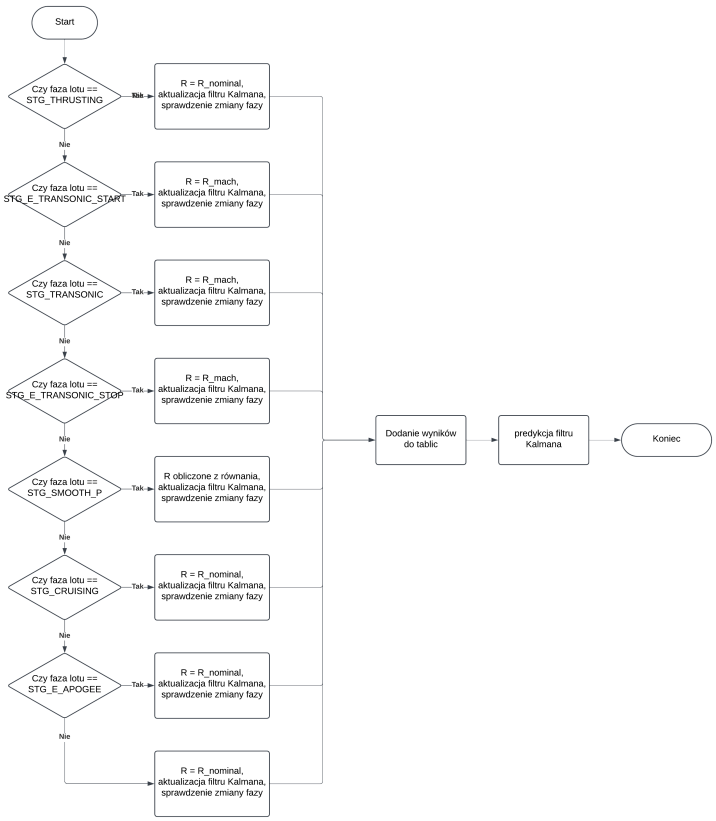
6.2.3 Wykrywanie faz lotu

W celu zapewnienia efektywnego działania algorytmu oraz spełnienia głównego zadania projektowego powstał algorytm zarządzający wykrywaniem faz lotu. Opiera się on na klasie składającej się z stopni, będących rozszerzeniem głównych faz opisanych w rozdziale 5.1.

```
class FlightStages_class():
def __init__(self):
    # initial flight stage
    self.flight_stage = self.STG_THRUSTING

    # time of flight events
    self.mach_start_time = None
    self.mach_end_time = None
    self.apogee_time = None

# flight phases
STG_THRUSTING           = 0
STG_E_TRANSONIC_START   = 1
STG_TRANSONIC           = 2
STG_E_TRANSONIC_STOP    = 3
STG_SMOOTH_P            = 4
STG_CRUISING            = 5
STG_E_APOGEE            = 6
STG_DESCENT             = 7
```

gdzie `flight_stage` to zmienna wewnętrzna przechowująca informację o aktualnej fazie lotu, natomiast `mach_start_time`, `mach_end_time`, `apogee_time` to zmienne przechowujące czas rozpo-

częcia danej fazy. Następnie, w klasie zdefiniowane zostały wartości numeryczne przypisane do konkretnego stopnia, które przyjmie zmienna `flight_stage`. Zawierają one tylko fazy i momenty lotu istotne z punktu widzenia algorytmu mającego na celu wykrycie apogeum lotu. Fazy trwające przez pewien czas zostały oznaczone prefiksem "STG", natomiast momenty rozdzielające fazy zostały określone prefiksem "STG_E". W tym miejscu należy zwrócić uwagę na rozszerzenie opisu lotu z prędkością zbliżoną lub powyżej prędkości dźwięku. Wyszczególniony jest moment wejścia w fazę `STG_E_TRANSONIC_START`, faza lotu określona została jako `STG_TRANSONIC`, zmienna `STG_E_TRANSONIC_END` określa moment wyjścia z fazy transoniki, natomiast `STG_SMOOTH_P` oznacza fazę lotu, której głównym celem jest stopniowe zmniejszenie wartości macierzy P.
Na rysunku 6.12 przedstawiony został schemat blokowy algorytmu wykrywania i zmiany faz.



Rysunek 6.12: Schemat blokowy algorytmu wykrywania faz lotu

Algorytm ten jest oparty o maszynę stanu, która sprawdza aktualną fazę lotu i wykonuje przypisaną jej instrukcję programu przypisując odpowiednią wartość do zmiennej R oraz wykonując równania aktualizacji filtru Kalmana. Następnie wartości wynikowe zapisywane są w tablicach i obliczane są równania predykcji.
Funkcje właściwe dla danych faz zostały zaimplementowane w głównej pętli programu:

```
for step in range(1, len(t_real)):
# manage flight stages for proper Kalman Update

if FlightStages.flight_stage == FlightStages.STG_THRUSTING:
    R = cfg.R_nominal
    alt_k, vel_k, P_t = KalmanFilter.update( pre_noise[step], R)
    FSTrans.FSTrans_THRUSTING( vel_k,
```

```
        cfg.start_mach_phase, cfg.mach_thres_sampl_nb, timestamp, FlightStages)

    elif FlightStages.flight_stage == FlightStages.STG_E_TRANSONIC_START:
        R = cfg.R_mach
        alt_k, vel_k, P_t = KalmanFilter.update( pre_noise[step], R)
        FSTrans.FSTrans_E_TRANSONIC_START( FlightStages )

    elif FlightStages.flight_stage == FlightStages.STG_TRANSONIC:
        R = cfg.R_mach
        alt_k, vel_k, P_t = KalmanFilter.update( pre_noise[step], R)
        FSTrans.FSTrans_TRANSONIC( vel_k, cfg.stop_transonic,
        cfg.mach_thres_sampl_nb, timestamp, FlightStages)

    elif FlightStages.flight_stage == FlightStages.STG_E_TRANSONIC_STOP:
        R = cfg.R_mach
        alt_k, vel_k, P_t = KalmanFilter.update( pre_noise[step], R)
        KalmanFilter.P = P_reset
        FSTrans.FSTrans_E_TRANSONIC_STOP( FlightStages )

    elif FlightStages.flight_stage == FlightStages.STG_SMOOTH_P:
        R = cfg.R_mach - (cfg.R_mach - cfg.R_nominal) /
        (FlightStages.mach_end_time+cfg.mach_smooth_P_time -
        FlightStages.mach_end_time) * (timestamp -
        FlightStages.mach_end_time)
        if R < cfg.R_nominal:
            R = cfg.R_nominal
        alt_k, vel_k, P_t = KalmanFilter.update( pre_noise[step], R)
        FSTrans.FSTrans_SMOOTH_P( vel_k,
        FlightStages.mach_end_time+cfg.mach_smooth_stage_time,
        cfg.mach_thres_sampl_nb, timestamp, FlightStages)

    elif FlightStages.flight_stage == FlightStages.STG_CRUISING:
        R = cfg.R_nominal
        alt_k, vel_k, P_t = KalmanFilter.update( pre_noise[step], R)
        FSTrans.FSTrans_CRUISING( vel_k, cfg.apogee_thres_sampl_nb,
        timestamp, FlightStages)

    elif FlightStages.flight_stage == FlightStages.STG_E_APOGEE:
        R = cfg.R_nominal
        alt_k, vel_k, P_t = KalmanFilter.update( pre_noise[step], R)
        FSTrans.FSTrans_E_APOGEE( FlightStages )

    elif FlightStages.flight_stage == FlightStages.STG_DESCENT:
        R = cfg.R_nominal
        alt_k, vel_k, P_t = KalmanFilter.update( pre_noise[step], R)

    kalman_alt.append( alt_k )
```

```
        kalman_vel.append( vel_k )
        R_plot.append( R )
        P_trace.append( P_t )

        #kalman prediction
        KalmanFilter.predict( acc_noise[step] )

        timestamp += Tp
```

gdzie `FlightStages` jest klasą zawierającą informacje o fazach lotu, `R` jest zmienną przechowującą aktualną wartość wariancji błędu pomiarowego, `cfg.R_nominal` oraz `cfg.R_mach` to wartości wariancji błędu pomiarowego odpowiednie dla fazy lotu poza transoniką i w trakcie transoniki, `KalmanFilter.update` to funkcja klasy filtru Kalmana realizująca równania aktualizacji dla EKF, natomiast `FSTrans` to klasa zawierająca algorytmy określające moment zmiany danej fazy lotu. Klasa `FlightStagesTransitions_class` zawiera algorytmy wykrywania poszczególnych faz lotu:

```
class FlightStagesTransitions_class:
def __init__(self):
    self.estim_mach_start_counter = 0
    self.estim_mach_end_counter = 0
    self.estim_apogee_counter = 0
    self.estim_trans_end_counter = 0

def FSTrans_THRUSTING(self, est_vel, start_mach_phase,
mach_thres_sampl_nb, timestamp, FlightStages):
    # conditions of phase end:
    if est_vel > start_mach_phase:
        self.estim_mach_start_counter += 1
        if self.estim_mach_start_counter == mach_thres_sampl_nb:
            FlightStages.mach_start_time = timestamp
            FlightStages.flight_stage = FlightStages.STG_E_TRANSONIC_START

def FSTrans_E_TRANSONIC_START(self, FlightStages):
    FlightStages.flight_stage = FlightStages.STG_TRANSONIC

def FSTrans_TRANSONIC(self, est_vel, v_stop_transonic,
mach_thres_sampl_nb, timestamp, FlightStages):
    # conditions of phase end:
    if est_vel < v_stop_transonic:
        self.estim_mach_end_counter += 1
        if self.estim_mach_end_counter == mach_thres_sampl_nb:
            FlightStages.mach_end_time = timestamp
            FlightStages.flight_stage = FlightStages.STG_E_TRANSONIC_STOP

def FSTrans_E_TRANSONIC_STOP(self, FlightStages):
    FlightStages.flight_stage = FlightStages.STG_SMOOTH_P
```

```
    def FSTrans_SMOOTH_P(self, est_vel, smooth_stage_stop_time,
    mach_thres_sampl_nb, timestamp, FlightStages):
        # conditions of phase end:
        if timestamp > smooth_stage_stop_time:
            self.estim_trans_end_counter += 1
            if self.estim_trans_end_counter == mach_thres_sampl_nb:
                FlightStages.flight_stage = FlightStages.STG_CRUISING

    def FSTrans_CRUISING(self, est_vel, apogee_thres_sampl_nb, timestamp,
    FlightStages):
        # conditions of phase end:
        if est_vel <= 0:
            self.estim_apogee_counter += 1
            if self.estim_apogee_counter == apogee_thres_sampl_nb:
                FlightStages.apogee_time = timestamp
                FlightStages.flight_stage = FlightStages.STG_E_APOGEE

    def FSTrans_E_APOGEE(self, FlightStages):
        FlightStages.flight_stage = FlightStages.STG_DESCENT
```

gdzie funkcja `__init__` definiuje liczniki będące buforem przy podejmowaniu decyzji o zmianie funkcji, a następnie zdefiniowane są algorytmy decydujące o zmianie danej fazy lotu.
Funkcje dotyczące końca faz `THRUSTING`, `TRANSONIC` i `CRUISING` oparte są na estymowanej prędkości rakiety i podejmują decyzję o zmianie fazy na podstawie przekroczenia odpowiedniego progu jej wartości, wydarzenia w trakcie lotu trwają jedną iterację głównej pętli programu, natomiast warunki zakończenia fazy `SMOOTH_P` zostały oparte o czas trwania funkcji i są opisane w następnej sekcji określonej numerem 6.2.4.

6.2.4 Dobór niepewności pomiarowej R

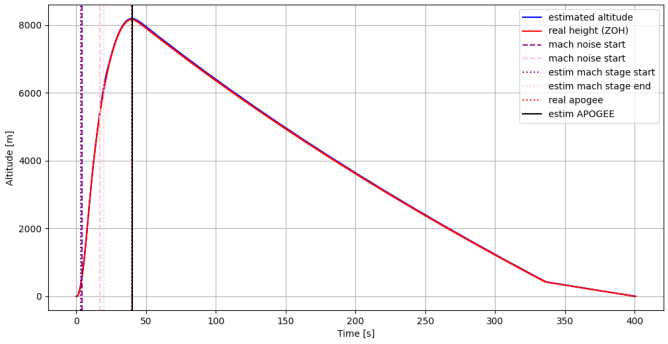
Faza oznaczona jako `SMOOTH_P` jest odpowiedzią na dużą zmianę estymowanej prędkości w momencie wyjścia z fazy transoniki opisanej w sekcji 6.1.7. W tym celu zaimplementowano liniową zmianę wartości kowariancji błędu pomiarowego R aby zniwelować nagłe zmniejszenie wartości śladu macierzy P powodujące natychmiastową aktualizację wysokości lotu i estymację dużej wartości prędkości. W pętli głównej programu, we fragmencie wywoływanym w trakcie fazy lotu `STG_SMOOTH_P` wywoływana jest funkcja oparta o wzór 6.6 [8].

$$R(t) = R_{\max} - \frac{R_{\max} - R_{\min}}{t_{\text{stop}} - t_{\text{start}}} \cdot (t - t_{\text{start}}) \tag{6.6}$$

gdzie $R_{\max}$ i $R_{\min}$ to maksymalna wartość wariancji R, t_{start} i t_{stop} to czas początku i końca zmiany liniowej wartości R, natomiast t to zmienna czasu.
Stwierdzono, że uzależnienie wzoru od czasu jest najlepszym podejściem, ponieważ alternatywą byłoby wykorzystanie estymowanej prędkości, jednak w sytuacji, gdy prędkość ta zostałaby obliczona w błędny sposób, wpłynęłoby to na wartość zmiennej R, sprawiając, że przestanie być liniowa. Ponadto, ze względu na mocne oscylacje wartości macierzy P w trakcie zmiany wartości kowariancji błędu pomiarowego R, trwanie całej fazy `SMOOTH_P` uzależniono od predefiniowanej wartości czasowej, ustawionej na stałe jako 7 sekund.

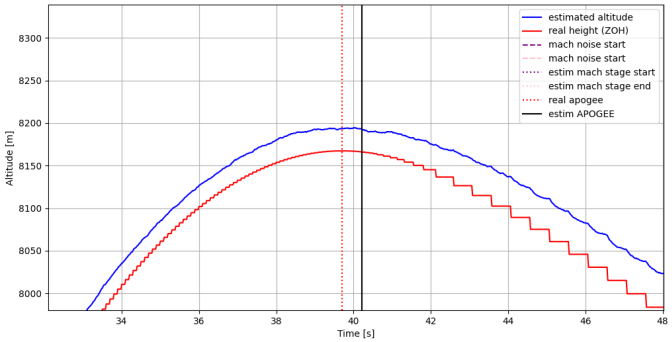
6.2.5 Wyniki, analiza i ewaluacja EKF

W tym podrozdziale przedstawione zostały wykresy zaimplementowanego algorytmu rozszerzonego filtru Kalmana oraz ich analiza i ewaluacja.



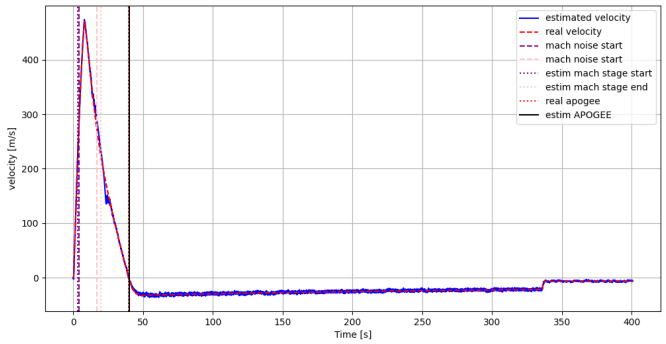
Rysunek 6.13: Wykres porównujący przebieg estymowanej i rzeczywistej wysokości lotu rakiety, z pionowymi liniami oznaczającymi momenty rozpoczęcia i końca fazy zwiększonego zaszumienia pomiarów oraz wykrycia faz lotu, w tym apogeum.

Na rysunku 6.13 przedstawiony został wykres wysokości lotu rakiety oraz momenty wykrycia faz lotu, w tym apogeum. W porównaniu do wykresów wygenerowanych z pomocą liniowego filtru Kalmana, nie zostały tutaj przedstawione obliczone wartości wysokości ze zmierzonej wartości ciśnienia ze względu na wprowadzenie tych równań wewnątrz filtru Kalmana. Wykres ten charakteryzuje się wysoką poprawnością i odwzorowaniem przebiegu rzeczywistego. Średnia wartość wskaźnika RMSE dla tego algorytmu wyniosi 27.9.



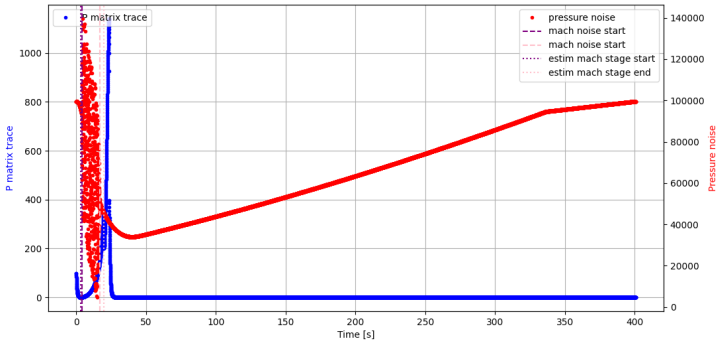
Rysunek 6.14: Fragment wykresu przedstawiającego przebieg rzeczywistej i estymowanej wartości wysokości lotu rakiety, przedstawiający rzeczywisty i wykryty moment apogeum rakiety.

Na wykresie 6.14 przedstawiony został fragment wykresu obrazującego przebieg estymowanej i rzeczywistej wartości wysokości lotu rakiety, zawierający oznaczone momenty rzeczywistego apogeum oraz jego wykrycia przez algorytm. Widoczna jest pewna różnica w odwzorowaniu przebiegu wysokości lotu, jednak mieści się ona w granicy tolerancji. Widoczna jest również pewna różnica pomiędzy apogeum rzeczywistym, a wykrytym przez algorytm, jednak ze względu na zastosowany mechanizm bezpieczeństwa wysyłający sygnał o apogeum dopiero po trzydziestu próbkach, w których niezmiennie występowała estymacja prędkości poniżej zera, pewne opóźnienie jest dozwolone. Ponieważ czas próbkowania wynosi 0.02 s, minimalna możliwa różnica wynosi 0.6 s, czyli dokładnie tyle ile pokazuje wykres, dzięki czemu spełniony jest warunek poprawnego wykrycia apogeum.



Rysunek 6.15: Wykres porównujący przebieg estymowanej i rzeczywistej prędkości lotu rakiety, z pionowymi liniami oznaczającymi momenty rozpoczęcia i końca fazy zwiększonego zaszumienia pomiarów oraz wykrycia faz lotu, w tym apogeum.

Rysunek 6.15 przedstawia wykres porównujący wartości prędkości rzeczywistej do wartości estymowanej za pomocą filtru. Z grafiki określić można, że prędkość odwzorowana jest z bardzo dużą dokładnością, ponadto widoczne w przypadku liniowego filtru Kalmana nagłe, gwałtowne i duże odchylenie prędkości w tym przypadku nie występuje. W momencie resetu macierzy, w miarę jak filtr zwiększa ufność pomiarów ciśnienia, pojawia się pewna zmiana, nie wpływa ona jednak na całokształt działania alogrytmu.



Rysunek 6.16: Wykres porównujący przebieg zmiany wartości śladu macierzy P oraz pomiarów ciśnienia.

Na rysunku 6.16 widnieje wykres porównujący przebieg zmiany śladu macierzy P oraz pomiarów ciśnienia. Można na nim zaobserwować to, że maksymalna wartość śladu macierzy P nie jest tak wielka, jak na wykresie przedstawionym na rysunku 6.11. Ponadto, dzięki liniowej zmianie wartości R opisanej w rozdziale 6.2.4, spadek następujący po resecie macierzy nie jest natychmiastowy i przez pewien czas uzyskuje wartości pośrednie.
Jak wykazano, zastosowane usprawnienia wprowadzone w algorytmie odpowiadają na problemy wykazane w ewaluacji filtru liniowego przedstawione w rozdziale 6.1.7, min. duże odchylenie estymowanej wysokości lotu od jej faktycznej wartości, gwałtowny spadek estymowanej wartości następujący w momencie aktualizacji oraz wysoka wartość śladu macierzy P z następującym natychmiastowym spadkiem do wartości nominalnej. Ponadto, wzięcie pod uwagę poprawnego określenia momentu apogeum lotu pozwala na stwierdzenie, że zaproponowany algorytm wykrywania parametrów lotu rakiety na podstawie odczytów barometru spełnia postawione przed nim zadania.

6.3 Implementacja algorytmu w języku C

W tym rozdziale została przedstawiona implementacja algorytmu w języku C. Ze względu na to, że komputer pokładowy rakiety jest w trakcie prac projektowych, pełna implementacja i testy algorytmu w środowisku właściwym dla urządzenia nie były możliwe. Z tego względu została stworzona propozycja algorytmów i funkcji w języku C++ i FreeRTOS, które, po zakończeniu prac nad komputerem, zostaną wdrożone.

6.3.1 Implementacja funkcji

W tym rozdziale zostały przedstawione najważniejsze algorytmy zaimplementowane w języku C i FreeRTOS.
Poniżej został przedstawiony algorytm filtru Kalmana:

```
    void KalmanFilter_Init(KalmanFilter *kf, float Tp, float p0, float alt0, float T0, float L)
    kf->F[0][0] = 1.0f;
    kf->F[0][1] = Tp;
    kf->F[1][0] = 0.0f;
    kf->F[1][1] = 1.0f;

    kf->G[0] = 0.5f * Tp * Tp;
    kf->G[1] = Tp;

    kf->Q[0][0] = ACC_VAR * Tp * Tp * Tp * Tp / 4.0f;
    kf->Q[0][1] = ACC_VAR * Tp * Tp * Tp / 2.0f;
    kf->Q[1][0] = ACC_VAR * Tp * Tp * Tp / 2.0f;
    kf->Q[1][1] = ACC_VAR * Tp * Tp;

    kf->P[0][0] = P_INIT_VALUE;
    kf->P[0][1] = 0.0f;
    kf->P[1][0] = 0.0f;
    kf->P[1][1] = P_INIT_VALUE;

    kf->x[0] = alt0;
    kf->x[1] = VELOCITY_START;

    kf->p0 = p0;
    kf->alt0 = alt0;
    kf->T_K = T0 + 273.15f;
    kf->L = L;
}

void KalmanFilter_Predict(KalmanFilter *kf, float u) {
    // Predict state
    kf->x[0] = kf->F[0][0] * kf->x[0] + kf->F[0][1] * kf->x[1] + kf->G[0] * u;
    kf->x[1] = kf->F[1][0] * kf->x[0] + kf->F[1][1] * kf->x[1] + kf->G[1] * u;
```

```
    // Predict covariance
    float P00 = kf->P[0][0], P01 = kf->P[0][1], P10 = kf->P[1][0], P11 = kf->P[1][1];
    kf->P[0][0] = kf->F[0][0] * P00 + kf->F[0][1] * P10 + kf->Q[0][0];
    kf->P[0][1] = kf->F[0][0] * P01 + kf->F[0][1] * P11 + kf->Q[0][1];
    kf->P[1][0] = kf->F[1][0] * P00 + kf->F[1][1] * P10 + kf->Q[1][0];
    kf->P[1][1] = kf->F[1][0] * P01 + kf->F[1][1] * P11 + kf->Q[1][1];
}

void KalmanFilter_Update(KalmanFilter *kf, float z, float R, float *alt_k, float *vel_k, float
    // Observation model
    float h_x = kf->p0 * powf(1.0f + (kf->L / kf->T_K) * (kf->x[0] - kf->alt0), 5.2558f);
    float dh_dx = 5.2558f * kf->L * kf->p0 * powf(fabsf((kf->L * (kf->x[0] + kf->alt0)) / kf->T

    // Kalman gain
    float K[2];
    float S = dh_dx * kf->P[0][0] * dh_dx + R;
    K[0] = kf->P[0][0] * dh_dx / S;
    K[1] = kf->P[1][0] * dh_dx / S;

    // Update state estimate
    float y = z - h_x; // Measurement residual
    kf->x[0] += K[0] * y;
    kf->x[1] += K[1] * y;

    // Update covariance
    float P00 = kf->P[0][0], P01 = kf->P[0][1], P10 = kf->P[1][0], P11 = kf->P[1][1];
    kf->P[0][0] = (1 - K[0] * dh_dx) * P00;
    kf->P[0][1] = (1 - K[0] * dh_dx) * P01;
    kf->P[1][0] = (1 - K[1] * dh_dx) * P10;
    kf->P[1][1] = (1 - K[1] * dh_dx) * P11;

    // Output results
    *alt_k = kf->x[0];
    *vel_k = kf->x[1];
    *P_trace = kf->P[0][0] + kf->P[1][1];
}
```

gdzie `KalmanFilter_Init` to funkcja inicjująca wartości początkowe filtru Kalmana, `KalmanFilter_Predict` to funkcja realizująca obliczenia predykcji filtru Kalmana, `KalmanFilter_Update` to funkcja realizująca obliczenia aktualizacji filtru Kalmana, natomiast`kf` to struktura zawierająca macierze filtru Kalmana.

Poniżej przedstawione zostały funkcje realizujące równania przejścia pomiędzy poszczególnymi fazami lotu rakiety:

```
    void FlightStages_Init(FlightStages *fs) {
    fs->flight_stage = STG_THRUSTING;
    fs->mach_start_time = 0.0f;
```

```
    fs->mach_end_time = 0.0f;
    fs->apogee_time = 0.0f;
}

void FlightStagesTransitions_Init(FlightStagesTransitions *fst) {
    fst->estim_mach_start_counter = 0;
    fst->estim_mach_end_counter = 0;
    fst->estim_apogee_counter = 0;
    fst->estim_trans_end_counter = 0;
}

void FSTrans_THRUSTING(FlightStagesTransitions *fst, FlightStages *fs, float est_vel, float sta
    if (est_vel > start_mach_phase) {
        fst->estim_mach_start_counter++;
        if (fst->estim_mach_start_counter >= mach_thres_sampl_nb) {
            fs->mach_start_time = timestamp;
            fs->flight_stage = STG_E_TRANSONIC_START;
        }
    }
}

void FSTrans_E_TRANSONIC_START(FlightStages *fs) {
    fs->flight_stage = STG_TRANSONIC;
}

void FSTrans_TRANSONIC(FlightStagesTransitions *fst, FlightStages *fs, float est_vel, float v_s
    if (est_vel < v_stop_transonic) {
        fst->estim_mach_end_counter++;
        if (fst->estim_mach_end_counter >= mach_thres_sampl_nb) {
            fs->mach_end_time = timestamp;
            fs->flight_stage = STG_E_TRANSONIC_STOP;
        }
    }
}

void FSTrans_E_TRANSONIC_STOP(FlightStages *fs) {
    fs->flight_stage = STG_SMOOTH_P;
}

void FSTrans_SMOOTH_P(FlightStagesTransitions *fst, FlightStages *fs, float timestamp, float sm
    if (timestamp > smooth_stage_stop_time) {
        fst->estim_trans_end_counter++;
        if (fst->estim_trans_end_counter >= mach_thres_sampl_nb) {
            fs->flight_stage = STG_CRUISING;
        }
    }
}
```

```
void FSTrans_CRUISING(FlightStagesTransitions *fst, FlightStages *fs, float est_vel, int apogee
    if (est_vel <= 0) {
        fst->estim_apogee_counter++;
        if (fst->estim_apogee_counter >= apogee_thres_sampl_nb) {
            fs->apogee_time = timestamp;
            fs->flight_stage = STG_E_APOGEE;
        }
    }
}

void FSTrans_E_APOGEE(FlightStages *fs) {
    fs->flight_stage = STG_DESCENT;
}
```

gdzie `FlightStages_Init` i `FlightStagesTransitions_Init` to funkcje inicjujące wartości własne algorytmu wykrycia zmian faz lotu takie jak flagi przechowujące czas zmiany oraz liczniki wykorzystywane jako bufor bezpieczeństwa przed wykryciem danej fazy, natomiast `fs` to struktura przechowująca informacje o aktualnej fazie lotu.

6.4 Perspektywy rozwoju algorytmu

Dzięki dokładnemu podejściu do zadania i zastosowaniu szeregu algorytmów spełniających różne funkcje, określono, że możliwa jest dalsza kontynuacja pracy nad algorytmem, dająca okazję do przystosowania go do spełnienia dodatkowych funkcji w systemach rakiety. Wprowadzenie algorytmu wykrywania faz lotu na podstawie zmian prędkości pozwala na rozszerzenie algorytmu w celu wykrywania dodatkowych stopni lotu takich jak start rakiety, wypalenie silnika lub upadek na powierzchni ziemi. Ponadto, dzięki zastosowaniu maszyny stanów realizującej w każdej fazie lotu inne funkcje, algorytm ten może stać się elementem pętli głównej komputera pokładowego, uruchamiającej wszystkie systemy rakiety zależne od danego stopnia lotu. Uwzględnienie w algorytmie warunków granicznych zakłóceń pomiaru ciśnienia sprawia, że algorytm jest bardzo odporny na wszelkie zakłócenia i możne być zastosowany do obsługi krytycznych systemów rakiety. Dodatkowo, dzięki bardzo dobrej estymacji wysokości lotu rakiety, dane wyjściowe filtru mogą zostać wykorzystane jako zapis przebiegu lotu w celu jego ewaluacji.

Rozdział 7

Podsumowanie

Proces tworzenia algorytmu doprowadził do powstania programu pozwalającego na trafne i odporne wykrywanie parametrów lotu rakiety, w tym przede wszystkim apogeum wysokości lotu, będącego jednym z najbardziej kluczowych momentów wystrzelenia rakiety. Rozpoczęto prace projektując algorytm w oparciu o liniowy filtr Kalmana, jednak okazał się posiadać niewystarczającą wydajność ze względu na nieliniowość przeliczania pomiaru na zmienną wektora stanu. Ponadto, po ulepszeniu funkcji szumującej i zastosowaniu symulacji błędu o rozkładzie jednostajnym, algorytm groził w niektórych wypadkach przedwczesnym, błędnym wykryciem apogeum rakiety, co mogłoby doprowadzić do uszkodzenia wystrzelonych spadochronów. Podjęto decyzję o ulepszeniu algorytmu i zastosowaniu rozszerzonego filtru Kalmana. Dodatkowo, zaimplementowano program wykrywający fazy lotu, dzięki czemu powstała możliwość wprowadzenia zmiennej wariancji błędu pomiarowego R pozwalającej na zmianę nastaw algorytmu w celu uniknięcia błędów pomiarowych w trakcie lotu z prędkością zbliżoną i większą niż prędkość dźwięku. To rozwiązanie uważane jest za jedno z bardziej profesjonalnych rozwiązań wśród twórców amatorskich zestawów rakietowych, którzy znacznie częściej decydują się na zastosowanie tzw. "mach lock", czyli ograniczenia czasowego, które ma za zadanie uruchomić wykrywanie apogeum lotu po określonym z symulacji czasie, po którym istnieje duża pewność, że rakieta znajduje się poza fazą transoniki.
Algorytm przyczynił się do rozwoju kompetencji i możliwości koła naukowego PUT Rocketlab i zostanie wykorzystany w trakcie najbliższego konkursu, w którym organizacja ma zamiar uczestniczyć. Ponadto, możliwości dalszego rozwoju algorytmu pozwalają na stworzenie nowego i ulepszonego algorytmu wykrywającego fazy lotu z dużą dokładnością i odpornością, dzięki czemu może zostać zastosowany w celu obsługi krytycznych systemów rakiety.

Literatura

[1] Alex Becker. *Kalman Filter from the Ground Up*. 2023.

[2] Tom Benson. Flight of a model rocket. `https://www.grc.nasa.gov/www/k-12/VirtualAero/BottleRocket/airplane/rktflight.html`, data skorzstania z źródła: 9.01.2025.

[3] Control and Telemetry Systems GmbH. Cats flight computer source code. `https://github.com/catsystems/cats-embedded`, data skorzystania z źródła: 11.01.2025.

[4] Mide Technology Corporation. Air pressure at altitude calculator. `https://www.mide.com/air-pressure-at-altitude-calculator`.

[5] B. Dołęga. *Struktura systemu układów pomiarowych samolotu tolerująca wybrane błędy ich działania*. Polskie Towarzystwo Mechaniki Teoretycznej i Stosowanej.

[6] Honeywell. *MPR-series MicroPressure sensor data sheet*.

[7] Jr John D. Anderson. Research in supersonic flight and the breaking of the sound barrier. `https://www.nasa.gov/history/SP-4219/Chapter3.html`.

[8] E. Kreyszig. *Advanced Engineering Mathematics*. Wiley, 2010.

[9] Jan Kędzierski. Filtr kalmana - zastosowania w prostych układach sensorycznych, 2007.

[10] Katedra Informatyki Stosowanej AGH Piotr Szwed. Eksploracja danych - ocena modeli, 2021.

[11] Bosch Sensortec. *BMI088 data sheet*.

[12] OpenRocket strona główna. `https://openrocket.info/`.

[13] yo3ict. Altitude pressure calculus - www.rocketryforum.com. `https://www.rocketryforum.com/threads/altitude-pressure-calculus.168590/`.

[14] Hoychoong Bang Youngjoo Kim. *Introduction to Kalman Filter and Its Applications*. Korea Advanced Institute of Science and Technology, 2018.

Instytut Robotyki i Inteligencji Maszynowej, Wydział Automatyki, Robotyki i Elektrotechniki
Politechnika Poznańska

Skład przy użyciu systemu LaTeX na platformie Overleaf.