POLITECHNIKA WROCŁAWSKA
WYDZIAŁ MECHANICZNY

KIERUNEK: MECHATRONIKA

MECHATRONIKA W POJAZDACH SAMOCHODOWYCH

Sprawozdanie: ABS

AUTOR:
Michał Synoś 205688
Mateusz Puzio 205677
Gr. Środa M07-03b 13^{15}

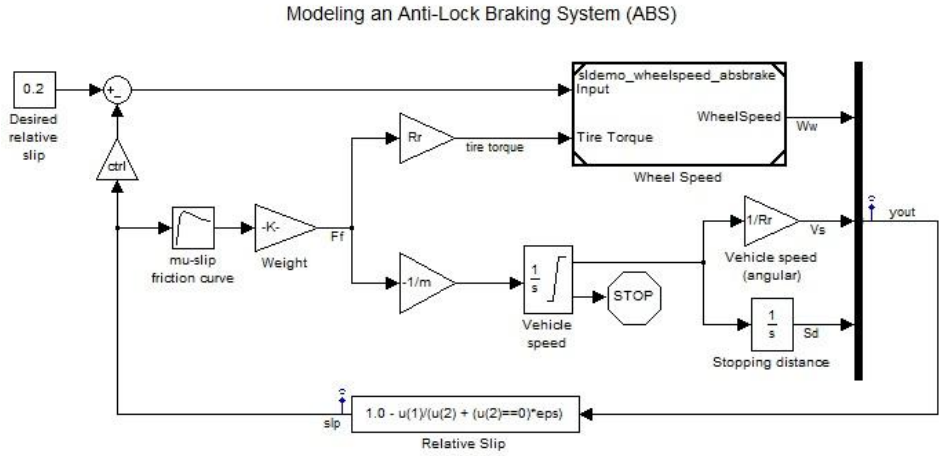
WROCŁAW 2016

1.) Wstep:

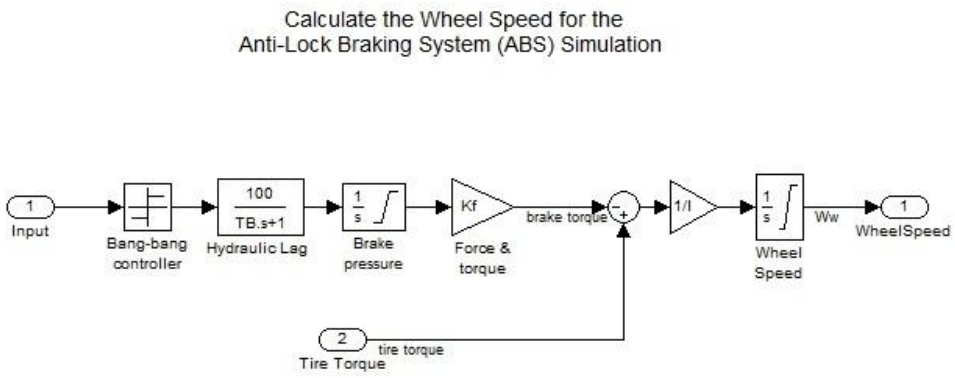
Aplikacja została wykonana zgodnie z założeniami przyjętymi wcześniej. Ma na celu symulację pracy układu ABS generując wykresy prędkości kół oraz samochodu, wartości poślizgu, momentu hamowania oraz drogi hamowania.

Największą trudnością w wykonaniu projektu okazało się określenie równań matematycznych dobrze reprezentujących model fizyczny układu ABS w środowisku C++. Początkowe dwa sposoby ()w tym z wykorzystaniem biblioteki boost) okazały się niewystarczająco dobre. Ostatecznie zadanie rozwiązano znając w sposób rozwiązania równań różniczkowych opisujących zmianę prędkości kątowej i liniowej w wyniku działania momentu oraz sił na koło samochodu. Dodatkowo znane jest rozwiązanie równania opisującego opóźnienie sygnały (Hydraulic LAG).

Sposób rozwiązania zadania nie był dowolny. Oparto się na modelu z biblioteki MatLAB, podążanie za modelem precyzuje sposób realizacji, co może utrudniać zadanie. Schematy z MatLAB'a przedstawiono na rysunkach 1.1 oraz 1.2.

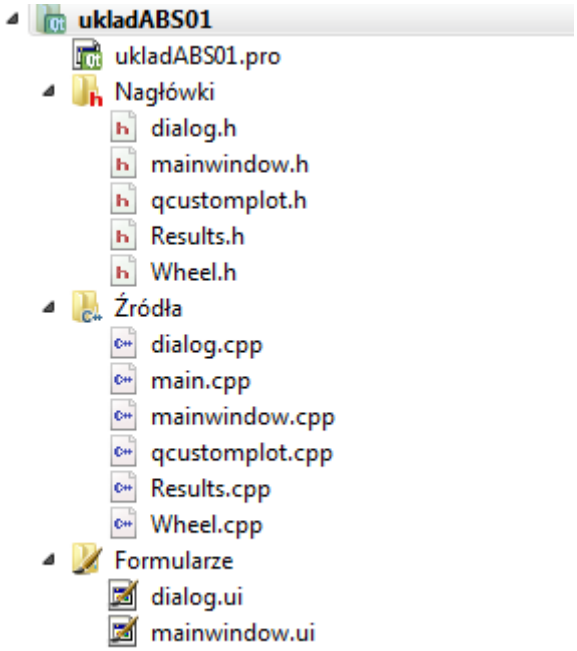


Rys.1.1 Schemat blokowy modelu systemu ABS



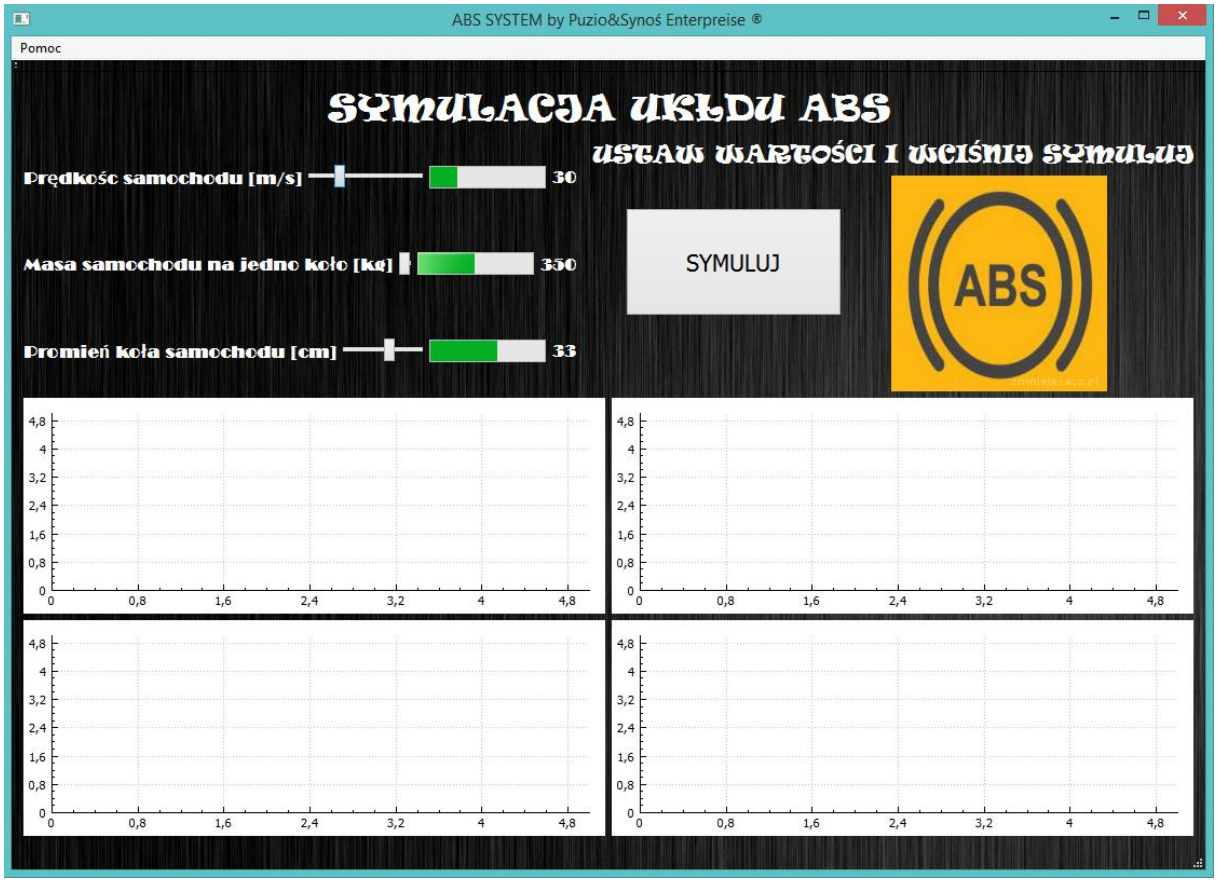
Rys.1.2 Schemat blokowy reprezentujący sposób wyliczania prędkości koła w modelu systemu ABS

Strukturę plików użytych w aplikacji przedstawiono na rysunku 1.3.

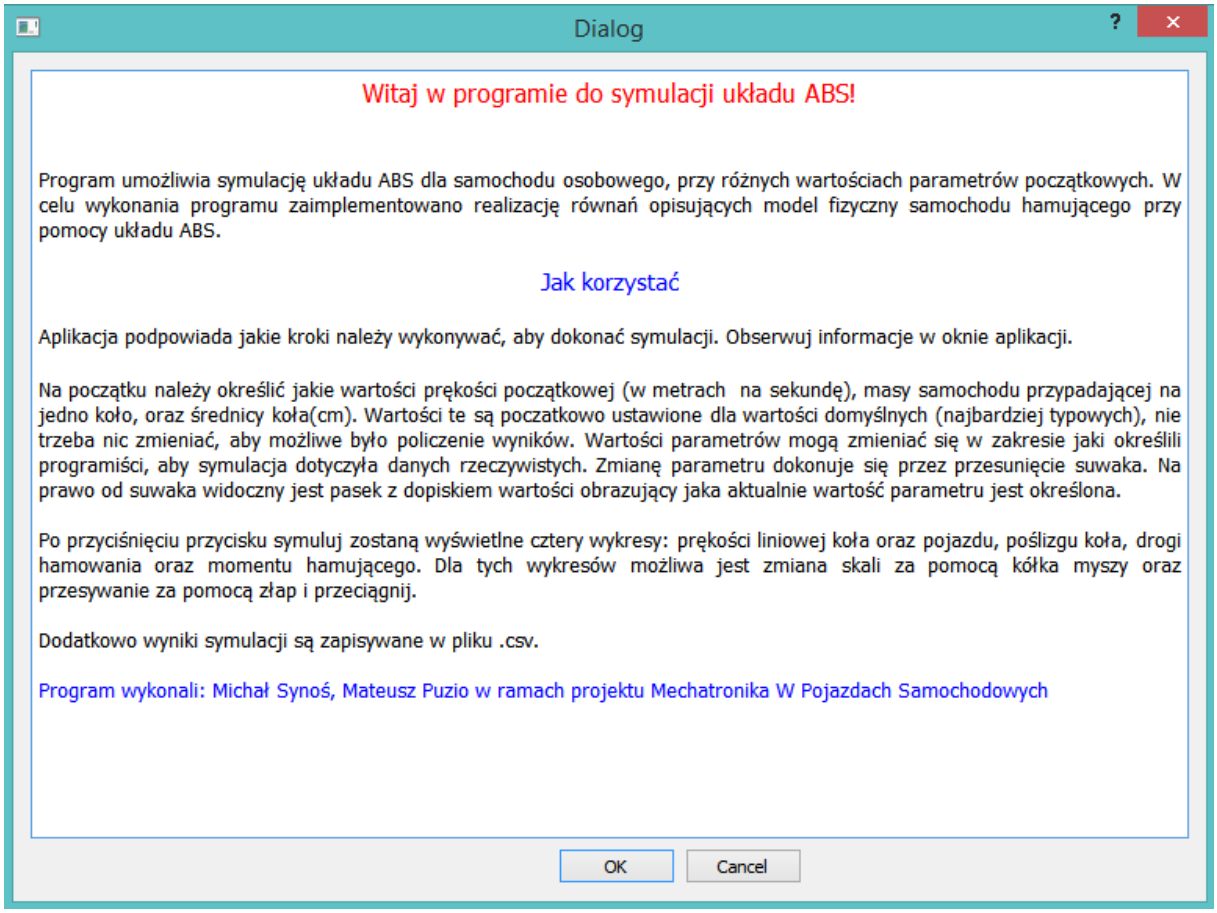


Rys. 1.3. Struktura plików aplikacji

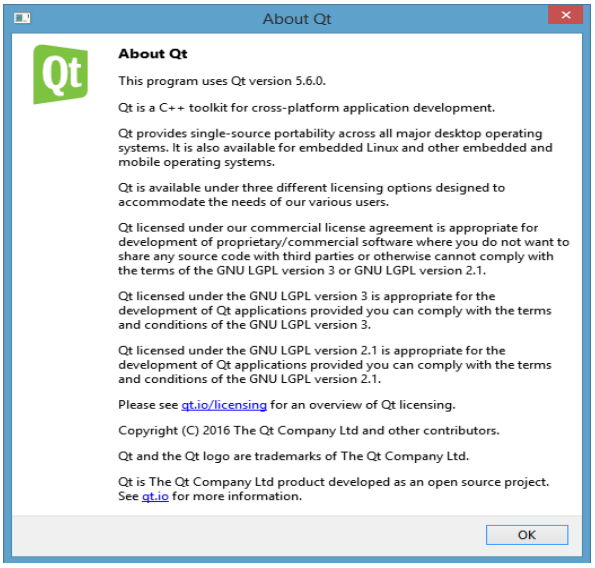
Na rysunkach 1.4., 1.5. oraz 1.6. przedstawiono wygląd uruchomionej aplikacji.



Rys. 1.2. Widok okna głównego aplikacji po włączeniu



Rys. 1.3. Widok okna pomocy jak korzystać z aplikacji



Rys. 1.4. Widok okna aboutQt

2.) Charakterystyka klasy Wheel:

Klasa Wheel jest odpowiedzialna za wykonywanie właściwych obliczeń systemu ABS. Posiada zaimplementowane metody obliczające chwilowe parametry które są kluczowe dla prawidłowego funkcjonowania systemu ABS. Na rysunku 2.1. przedstawiono wykaz zaimplementowane metody i wykorzystywane atrybuty z podziałem na dostęp private oraz public.

```
class Wheel
{
public:
    double vehicleSpeed = 15 ;
    double vehicleWeight = 350;
    double diameterOfCircle = 0.33;
    double torqueBraking = 35;
    Results results;
    Wheel();
    ~Wheel();
    QString doCalculations();
    void readDataFriction(QString location);
private:
    QVector<double> tableFrictionData;
    double samplingFrequency = 0.01;
    double gravitation = 9.81; // m/s^2
    double actSlip = 0;
    double momentOfInertia = 1.33;
    double vehicleAcceleration;
    double wheelAcceleration;
    double wheelSpeed = (vehicleSpeed / diameterOfCircle);
    double tireTorque = 0;
    double distanceToStop;
    int sign(double speedWheel);
    void calculationTorqueAcceleration();
    double integrationWheelSpeed();
    double integrationVehicleSpeed();
    void calculationSlip();
    double choiceFriction(double slip);
    void kutrapaliFunction();
    void bangBang();
    double absControl();
};
```

Rys. 2.1. Elementy składowe klasy Wheel

Użytkownik programu ma dostęp do parametrów wpływających na model hamowania jak: prędkość początkowa, średnica koła, czy wartość ciężaru przypadającego na koło podczas hamowania. Pozostałe parametry są zdefiniowane globalnie i użytkowni nie ma do nich dostępu. Są to : stała grawitacji, przybliżona wartość momentu bezwładności koła, czy częstotliwość próbkowania. Po kilku próbach doświadczalnych częstotliwość została dopasowana na najbardziej optymalnym poziomie, który zapewnia sprawny przebieg

obliczeń(nie zamula) oraz umożliwia uzyskanie przejrzystych wyników. Oprócz tego zdefiniowano początkową siłę hamowania z jaką system ma zastosować na początku procesu wyhamowywania pojazdu(zmienna torqueBraking). Klasa posiada własny zdefiniowany atrybut results utworzonej klasy Results, która przechowuje wyniki obliczeń dla danych podanych przez użytwkonika. Jako zmienne globalne są również przechowywane parametry ulegające zmianie w sposób iteracyjny co kolejny takt hamowania , oddzielony o częstotliwość próbkowania procesu hamowania. Do poprawnego działania programu konieczne jest wczytanie informacji o zależności współczynnika tarcia od wartości poślizgu. Program przeprowadza ten proces nie jawnie. Bardziej zaawansowany użytkownik może ręcznie podmienić te wartość ale nie istnieje taka potrzeba. Za wczytanie danych odpowiada funkcja readDataFriction(). Proces przeprowadzenia obliczeń dla danego kroku został podzielony na krytyczne fragmenty, co ułatwiło proces testowania programu, oraz poszukiwania ewentualnych błędów. Algorytmy zaimplementowane w metodach odpowiadają rozwiązaniu danego równania różniczkowego lub całkowego które jest konieczne do przeprowadzenia w danym etapie wykonania obliczeń.

Pierwszym etapem obliczeń jest wyznaczenie chwilowych wartości przyspieszeń działających na pojazd oraz na koło na podstawie sił działających na pojazd. Realizuje to funkcja calculationTorqueAccleration. Postać metody przedstawiono na rysunku 2.2.

```
19  void Wheel::calculationTorqueAcceleration() {
20      double Ftx = -(choiceFriction(actSlip)) * vehicleWeight * gravitation;
21      double torque = tireTorque - sign(wheelSpeed) * torqueBraking;
22      vehicleAcceleration = (Ftx / vehicleWeight);
23      wheelAcceleration = torque / momentOfInertia;
24  }
```

Rys. 2.2. Obliczanie przyspieszeń pojazdu ora koła

Podczas pierwszego kroku brany jest aktualna wartość poślizgu z początku hamowania. Zmienna actSlip przechowuje chwilową wartość poślizgu. Metoda choiceFriction zwraca wartość współczynnika tarcia na podstawie aktualnego poślizgu. Zależność tą wybierane na podstawie danych wczytanych przez program.

Kolejnym etapem jest wyznaczenie aktualnych wartości prędkości pojazdu na podstawie znajomości wartości tych prędkości w kroku poprzednim oraz wartości przyspieszeń w aktualnie rozpatrywanym kroku. W metodach realizujących to zadanie wykorzystanie rozwiązanie całkowania. Funkcje przedstawiono na rysunku 2.3.

```
26  double Wheel::integrationWheelSpeed() {
27      double tempWheelSpeed;
28      tempWheelSpeed = wheelSpeed + wheelAcceleration * samplingFrequency;
29      wheelSpeed = tempWheelSpeed;
30      if (wheelSpeed < 0) wheelSpeed = 0;
31      double romega = wheelSpeed * diameterOfCircle;
32      return romega;
33  }
34
35  double Wheel::integrationVehicleSpeed() {
36      double tempVehicleSpeed;
37      tempVehicleSpeed = vehicleSpeed + vehicleAcceleration * samplingFrequency;
38      return tempVehicleSpeed;
39  }
```

Rys. 2.3. Wyznaczenie chwilowych wartości prędkości pojazdu oraz koła

Obie funkcje zwracają obliczone wartości. Są one wywoływane w metodzie odpowiedzialnej za obliczenie chwilowej wartości współczynnika poślizgu calculationSlip. Postać tej metody przedstawiono na rysunku 2.4.

```
41  void Wheel::calculationSlip() {
42      double tempSpeedVehicle;
43      double romega = integrationWheelSpeed();
44      vehicleSpeed = integrationVehicleSpeed();
45      if (romega > vehicleSpeed) tempSpeedVehicle = romega;
46      else tempSpeedVehicle = vehicleSpeed;
47      actSlip = (romega - vehicleSpeed) / tempSpeedVehicle;
48      if (actSlip > 0) { actSlip = 0; }
49      distanceToStop = distanceToStop + vehicleSpeed * samplingFrequency;
50      results.wheelSpeedSave(romega);
51      results.vehicleSpeedSave(vehicleSpeed);
52      results.slipDataSave(actSlip);
53      results.distanceSave(distanceToStop);
54  }
```

Rys. 2.4. Wyznaczenie chwilowej wartości współczynnika poślizgu

Dodatkowym zadaniem tej metody jest wyznaczanie pokonanej drogi hamowania od momentu rozpoczęcia procesu oraz zapisanie tej wartości wraz z chwilową wartością prędkości i poślizgu do wyników. W ten sposób jesteśmy stanie zachować chwilowe wartości parametrów do późniejszej analizy.

Na tym etapie można zakończyć działanie programu dla danego kroku. W efekcie otrzymamy system symulujący hamowanie pojazdu bez systemu ABS z maksymalną siłą hamowania zdefiniowana na początku działania symulacji. Aby rozwinąć funkcjonalność systemu do sprawności ABS konieczne jest przeprowadzenie dodatkowych obliczeń. Pierwszą czynnością jest wyznaczenie chwilowej zmiany momentu działającego na koło spowodowane hamowaniem. Czynność ta realizowana jest w metodzie kutrapaliFunction(*easter egg*). Konieczne jest wprowadzenie kontrolera który będzie odzwierciedlał działanie fizycznego regulatora ciśnienia płynu hamulcowego w układzie hamulcowym. Jednocześnie wprowadzono

opóźnienie podjęcia decyzji w postaci przedziału różnic wartości poślizgu, w którym program nie zmienia siły hamowania. Metoda ta to bangBang(). Postać obu funkcji przedstawiono na rysunku 2.5.

```
97  void Wheel::kutrapaliFunction() {
98      double mu = choiceFriction(actSlip);
99      double tempTorque;
100     tempTorque = fabs(mu * diameterOfCircle * gravitation * vehicleWeight);
101     if ((tempTorque - tireTorque) > 18) tireTorque += 18;
102     else if ((tempTorque - tireTorque) < 18) tireTorque -= 18;
103     else (tireTorque = tempTorque);
104 }
105
106 void Wheel::bangBang() {
107     double slipError = absControl();
108     if (slipError > 0.017) {
109         torqueBraking = torqueBraking - 20;
110     } else
111         if (slipError < 0.017) {
112             torqueBraking = torqueBraking + 20;
113         }
114     results.torqueBrakingSave(torqueBraking);
115 }
```

Rys. 2.5. Wyznaczenie siły hamowania w następnym kroku

Różnica pomiędzy wartościami poślizgu w danym kroku a najbardziej optymalną wartością poślizgu jest obliczana metodą absControl(). Postać funkcji zaprezentowano na rysunku 2.6.

```
117 double Wheel::absControl() {
118     double desiredSlip = -0.2;
119     return (desiredSlip - actSlip);
120 }
```

Rys. 2.6. Obliczenie uchybu wartości poślizgu

Najlepsze parametry hamowania system uzyskuje przy wartości poślizgu 0,2, dla której otrzymamy najwyższą wartość współczynnika tarcia. Obliczenia są przeprowadzana do momentu gdy prędkość pojazdu nie spadnie poniżej ustalonej granicy. Na rysunku 2.7. przedstawiono główną metodę wywoływana z poziomu użytkownika. Odpowiada ona za wykonanie wszystkich obliczeń oraz udzielenie informacji o zakończeniu obliczeń.

```
122 QString Wheel::doCalculations() {
123     if (vehicleSpeed == 0 && vehicleWeight == 0 && diameterOfCircle == 0) {
124         return "Not specified the required parameters!!";
125     } else {
126         while (wheelSpeed) {
127             calculationTorqueAcceleration();
128             calculationSlip();
129             kutrapaliFunction();
130             bangBang();
131         }
132         return "Calculations were made!!";
133     }
134 }
```

Rys. 2.7. Metoda wywołująca kolejne kroki w algorytmie obliczeń

3.) Charakterystyka klasy Results:

Klasa Results definiuje typ obiektów w których mogą być przechowywane wyniki symulacji. Posiada ona zdefiniowane atrybuty w postaci wektorów mogących przechowywać wyniki symulacji dla poszczególnych parametrów, które są brane pod uwagę przy analizie poprawności działania systemu ABS. Na rysunku 3.1. zaprezentowano strukturę omawianej klasy Results.

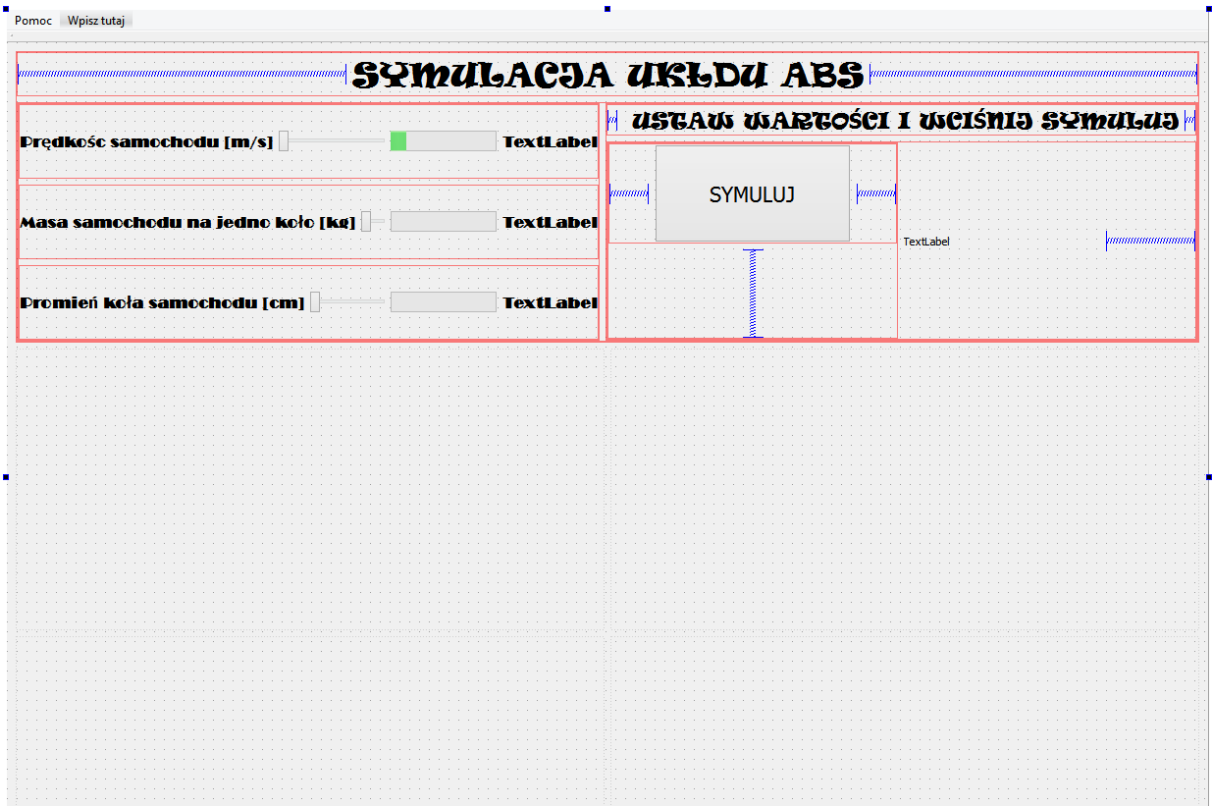
```
 7 class Results
 8 {
 9 public:
10     Results();
11     ~Results();
12     void saveResult(QString location);
13     void distanceSave(double distance);
14     void torqueBrakingSave(double brakingTorque);
15     void slipDataSave(double slip);
16     void wheelSpeedSave(double wheelSpeed);
17     void vehicleSpeedSave(double vehicleSpeed);
18     QVector<double> brakingTorqueData;
19     QVector<double> stopDistanceData;
20     QVector<double> slipData;
21     QVector<double> wheelSpeedData;
22     QVector<double> vehicleSpeedData;
23 };
```

Rys. 3.1. Wewnętrzna struktura klasy Results

Dodatkową metodą zdefiniowaną w klasie jest możliwość zapisu wyników symulacji do pliku.

4.) QtWidgets:

Rysunek 4.1. pokazuje rozmieszczenie elementów interfejsu. W celu prawidłowego skalowania się okna aplikacji użyto layout'ów oraz spacer'ów. Dzięki temu rozmiar okna można dynamicznie zmieniać, elementy przystosowują do tego swój rozmiar. Dla niektórych z nich zostały dodatkowo określone wymiary minimalne.



Rys. 4.1. Rozmieszczenie elementów interfejsu

Rysunek 4.2. prezentuje, co zastaje wykonane, gdy program wykryje zmianę wartości na suwaku odpowiedzialnym za ustawianie prędkości. Zostaje zmieniona wartość *progresssBar,* zmiana opisu *label* aby użytkownik widział zmianę liczbową, oraz, co najważniejsze, zostaje zmieniona wartość prędkości samochodu.

```
65  void MainWindow::on_horizontalSlider_PredkoscSamochodu_valueChanged(int value)
66  {
67
68      ui->progressBar_PredkoscSamochodu->setValue(value);
69      ui->label_PredkoscSamochoduValue->setText(QString::number(value));
70      predkoscSamochodu=(double)value;
71  }
```

Rys. 4.2. Działanie aplikacji po przesunięciu suwaka prędkości

W ramach prezentacji wykresów skorzystano z bibliotek QCustomPlot. Dzięki temu możliwe jest generowanie dynamicznych wykresów, możliwa jest interakcja użytkownika poprzez myszkę komputera i np. zmiana skali, przesunięcie. Na rysunku 4.3 zaprezentowano kod generujący wykres dla momentu obrotowego. Jak widać z tego rysunku, generowany jest nowy graf poprzez *AddGraph* po czym dodana jest linia wykresu poprzez *graph(0).* Oczywiście można dodać więcej linii na jeden wykres. Dane są wczytywane za pomocą *setData*, która przyjuje QVector na oś x oraz y. Ważne dla prawidłowego działania jest skalowanie obrazu *rescaleAxes(),* oraz, aby za każdym kolejnym naciśnięciem przycisku wykres został odświżony *replot().*

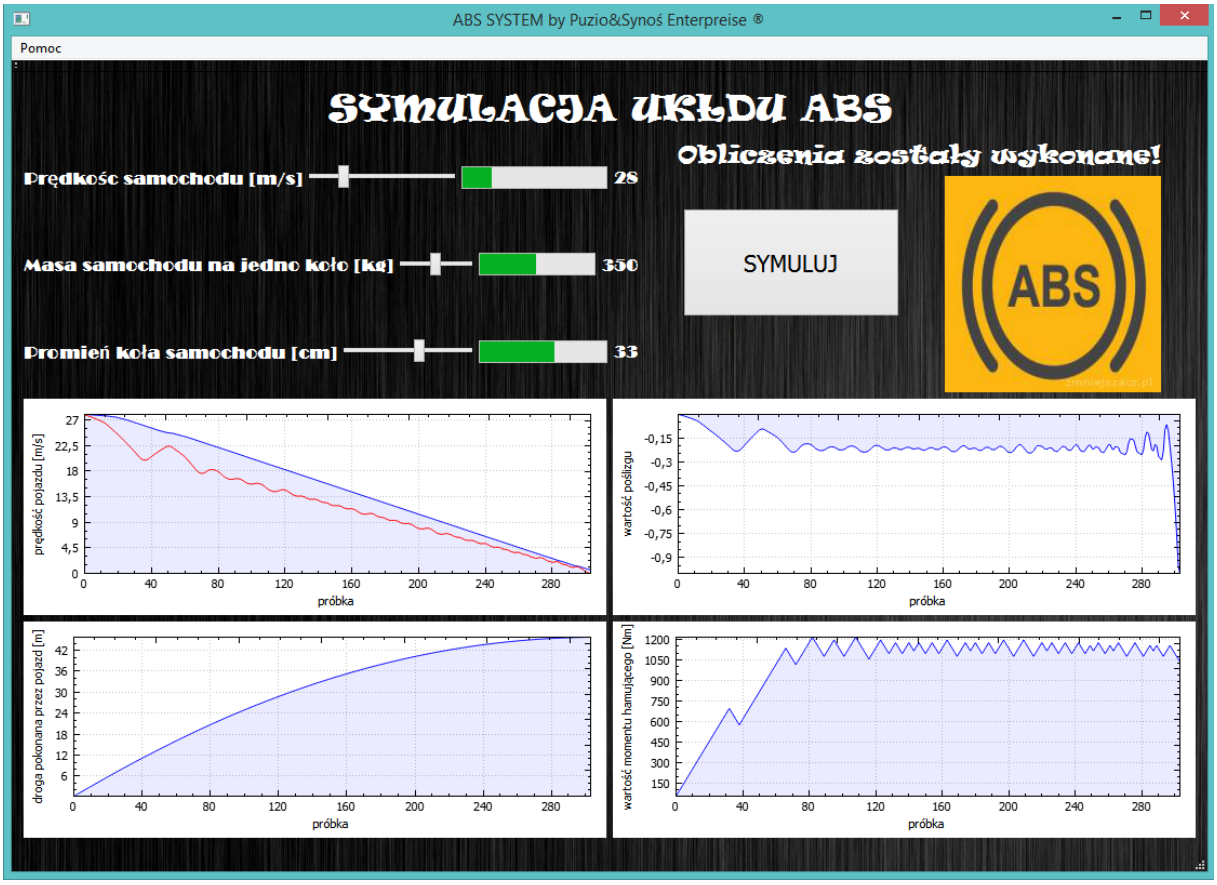
```
179         //WYKRES MOMENTU
180         ui->customPlot4->addGraph();
181         ui->customPlot4->graph(0)->setPen(QPen(Qt::blue));
182         ui->customPlot4->graph(0)->setBrush(QBrush(QColor(0, 0, 255, 20)));
183         ui->customPlot4->xAxis2->setVisible(true);
184         ui->customPlot4->xAxis2->setTickLabels(false);
185         ui->customPlot4->yAxis2->setVisible(true);
186         ui->customPlot4->yAxis2->setTickLabels(false);
187         ui->customPlot4->xAxis->setLabel("próbka");
188         ui->customPlot4->yAxis->setLabel("wartość momentu hamującego [Nm]");
189         ui->customPlot4->graph(0)->setData(x, test.results.brakingTorqueData);
190         ui->customPlot4->setInteractions(QCP::iRangeDrag | QCP::iRangeZoom | QCP::iSelectPlottables);
191         ui->customPlot4->rescaleAxes();
192         ui->customPlot4->replot();
```

Rys. 4.2. Generowanie wykresu momentu hamującego za pomocą QCustomPlot

5.) Przykład działania:

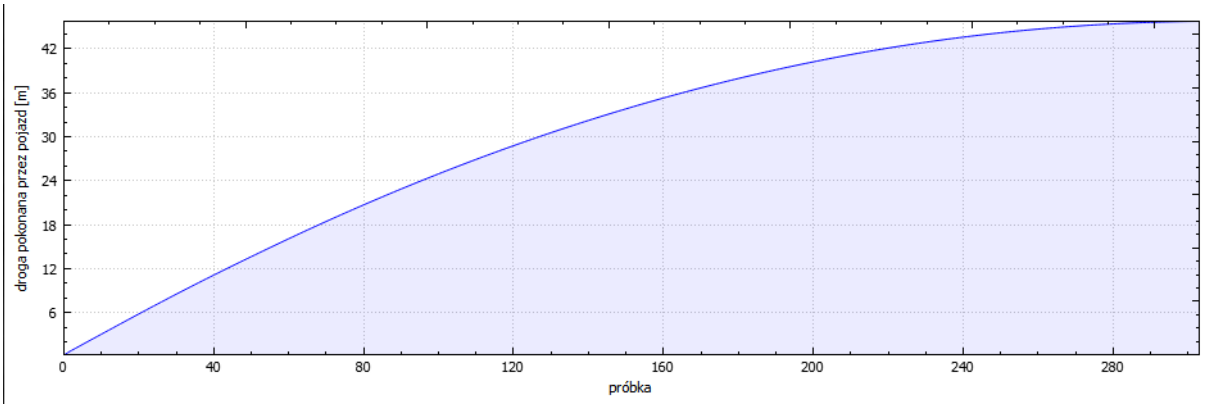
W aplikacji za pomocą suwaków można zmieniać parametry: promienia koła, masy samochodu na koło oraz prędkość początkową. Ta ostatnia ma oczywiście decydujący wpływ na wynik obliczeń. Jak wspomniano wcześniej aplikacja generuje szereg wykresów reprezentujących wyniki symulacji. W celu pokazania przykładowego działania aplikacji przeprowadzono wynik symulacji dla prędkości początkowej 28m/s (co daje około 100km/h) oraz dla prędkości dwa razy większej.

Na rysunku 5.1 przedstawiono przykładowy wynik działania dla 28m/s. Dalej zostanie przeprowadzona analiza porównawcza wyników dla poszczególnych wykresów.

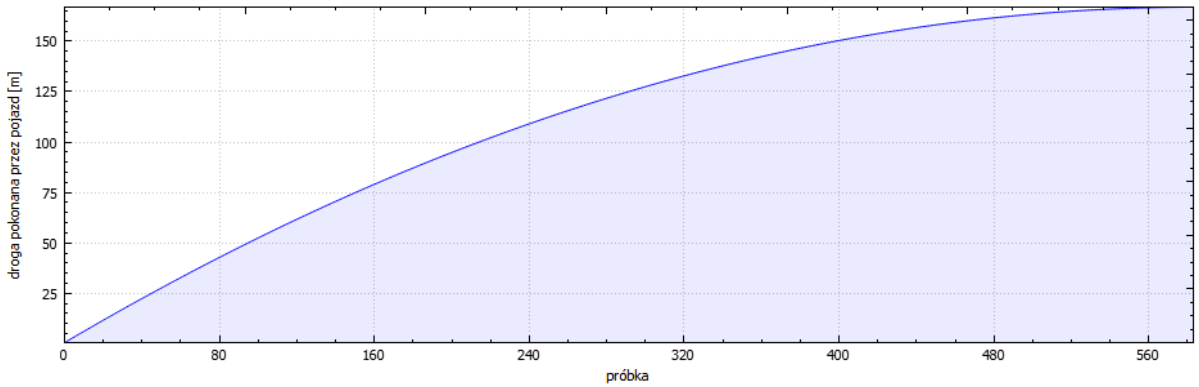


Rys. 5.1. Wynik działania aplikacji dla standardowych masie samochodu oraz promieniu koła. Wartość prędkości początkowej ustawiona na 28m/s.

Jak widać dla standardowej prękości 100km/h pojazd potrzebował około 42 metrów na zahamowanie do zera. Jest to wynik w zupełności pokrywający się z rzeczywistymi wynikami samochodów osobowych. Dla prędkości 56m/s spodziewamy się znacznego wzrostu drogi, co pokazują rysunki 5.2 oraz 5.3.

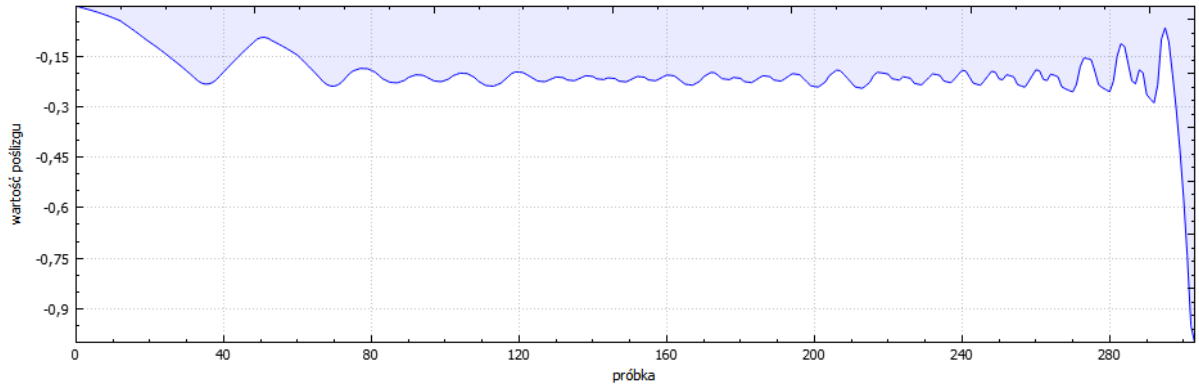


Rys. 5.2. Przy prędkości 28m/s droga hamowania wynosi ok 42 metrów.

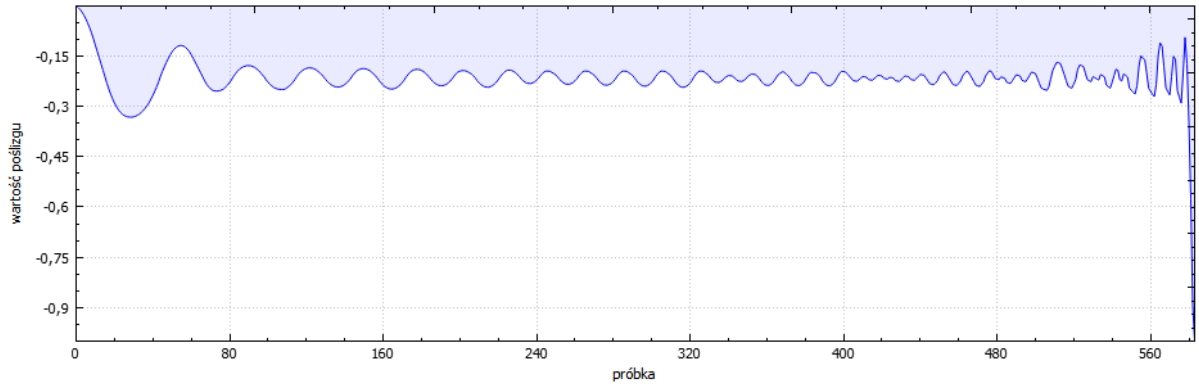


Rys. 5.3. Przy prędkości 56m/s droga hamowania wynosi ok 163 metry.

Rysunki 5.4 oraz 5.5 prezentują poślizg kół w czasie dla obu założonych wartości prędkości. Widać, że wynik oscyluje wokół wartości optymalnej tzn. poślizgu równego 0.2. Oczywiście im dłuższa droga hamowania tym więcej „do zrobienia" ma układ.



Rys. 5.4. Sterowanie wartością poślizgu przez układ ABS dla prędkości początkowej równej 28m/s



Rys. 5.5. Sterowanie wartością poślizgu przez układ ABS dla prędkości początkowej równej 56m/s

6.) Podsumowanie:

Podsumowując stwierdza się, że aplikacja, choć ograniczona, ma pewny wymiar praktyczny. Można by na przykład prezentować wyniki jej pracy na kursie na prawo jazdy, ukazując jak silnie rośnie droga hamowania wraz ze wzrostem wartości prędkości początkowej, oraz jak działa układ ABS. Dzięki tej wiedzy możliwe jest zwiększenie bezpieczeństwa na drodze. Program stanowi dobrą platformę rozwojową do budowy bardziej rozbudowanego sytemu symulacji pojazdu podczas nietypowych sytuacji. Warto podkreślić iż dokładność obliczeń oraz sensowność wyników silnie zależą od częstotliwości próbkowania. Przy zbyt dużej częstotliwości układ zostanie „zasypany" przez wyniki obliczeń cząstkowych, co może spowodować zawieszenie działania programu. Przy zbyt małej częstotliwości próbkowania program nie będzie w stanie podjąć najbardziej optymalną decyzje w danej chwili z powodu szybszej reakcji układu. Dokładność symulacji zależy również od wielkości przedziału uchybu wartości poślizgu w którym „sterownik" ABS nie zmienia wartości momentu hamującego. Jesteśmy w stanie zamodelować układ ABS który bezzwłocznie zareaguje na zmianę wartości współczynnika poślizgu lecz będzie on daleki od układów stosowanych w pojazdach. Sam regulator zamodelowany w programie jest typowym regulatorem proporcjonalnym. O wiele lepszym rozwiązaniem jest zastosowanie regulatora PID. Przy odpowiednio dobranych parametrach nastawach regulatora otrzymamy najlepszy efekt wynikający z hamowania przy wykorzystaniu systemu ABS. Prace teoretyczne nad tego typu układami są prowadzone.

W aplikacji zaimplementowano matematyczny model skomplikowanego układu fizycznego. Dodatkowo ładna i estetyczna prezentacja okienkowa zachęca do pracy z aplikacją, bardziej, niż chociażby aplikacja konsolowa.

Stwierdza się, że projekt można określić jako udany, założenia projektowe zostały zrealizowane(dla jednego koła, dla czterech kół wynik nie zmieni się).