ArduinoControl

[Dokumentacja techniczna]

Technologia: Java
Środowisko: Android Studio
API level: 15
Android 4.0.3 (IceCreamSandwich)

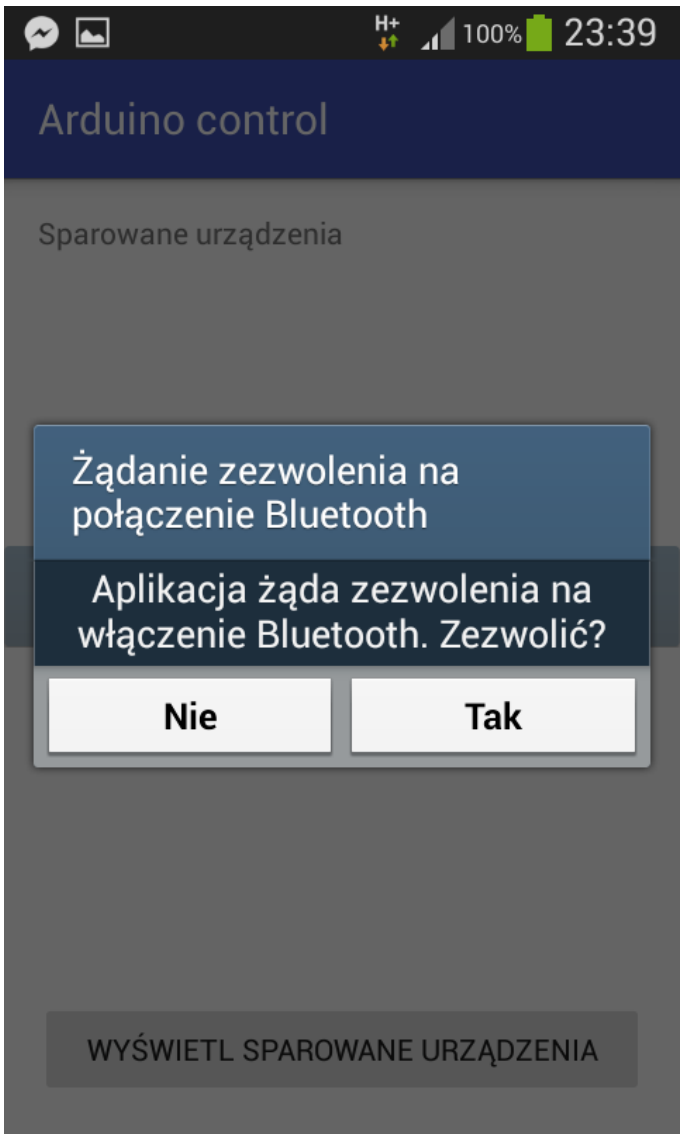
ArduinoControl
Aplikacja na platformę Android wysyłająca komunikaty drogą radiową (**Bluetooth)** służąca do kontroli robota opartego o płytkę Arduino Genuino Uno z modułem bluetooth HC-06.

Użytkownik
Jest to osoba, która chce za pomocą aplikacji na androida połączyć zdalnie z robotem poprzez Bluetooth i wydawać mu polecenia.

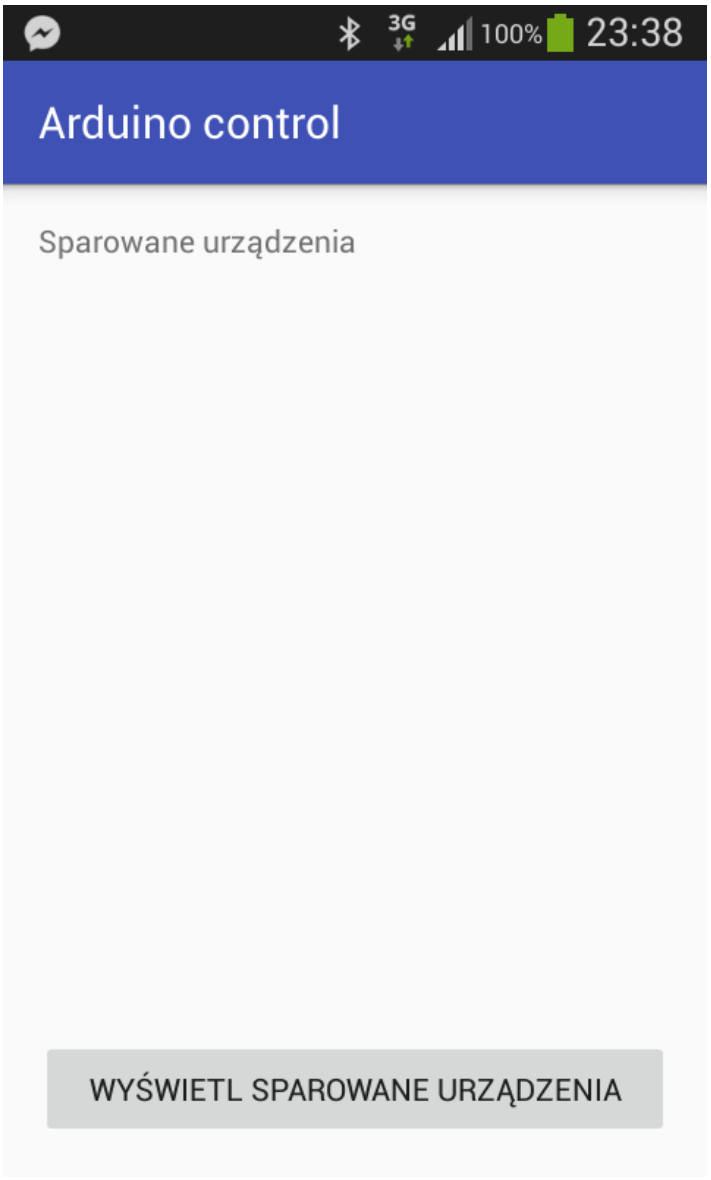
Przypadek użycia: “Zdalne sterowanie robotem”
Użytkownik korzystając z aplikacji łączy się z wcześniej sparowanym urządzeniem i steruje nim. Z dostępnych metod sterowania może wybrać sterowanie manualne bądź za pomocą jednego z dwóch algorytmów: używając czujników linii (line tracking) albo czujników zbliżeniowych (ultrasonic).

Aplikacja wykorzystuje protokół SPP (profil K5)
Aplikacja łączy się z drugim urządzeniem obsługującym profil **SPP(K5) –** profil wirtualnego portu szeregowego SPP (Serial Port Profile) Profil portu szeregowego opisuje wymagania związane z realizacją emulowanego radiowego łącza szeregowego np. pomiędzy dwoma komputerami. Umożliwia wysyłanie pakietów danych pomiędzy dwoma urządzeniami

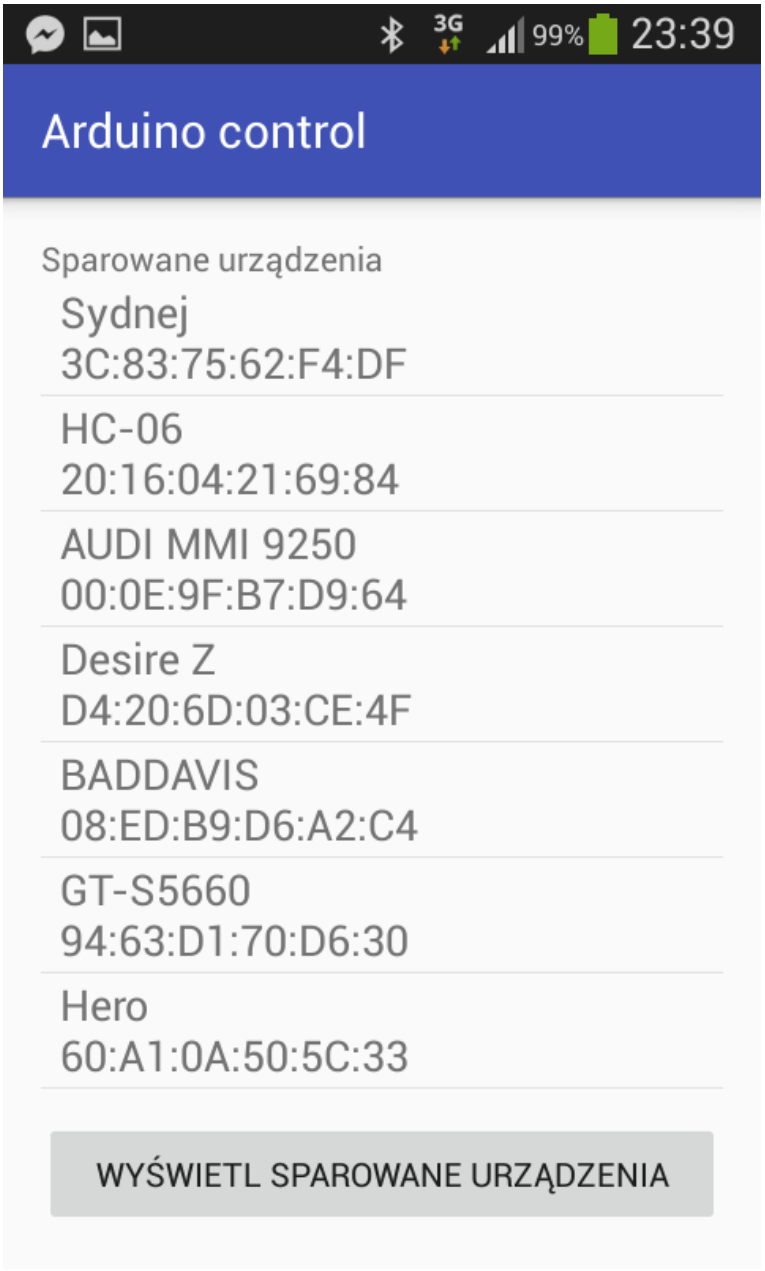
Ekran główny z zapytaniem o włączenie bluetooth:



Ekran główny przed wyświetleniem sparowanych urządzeń:

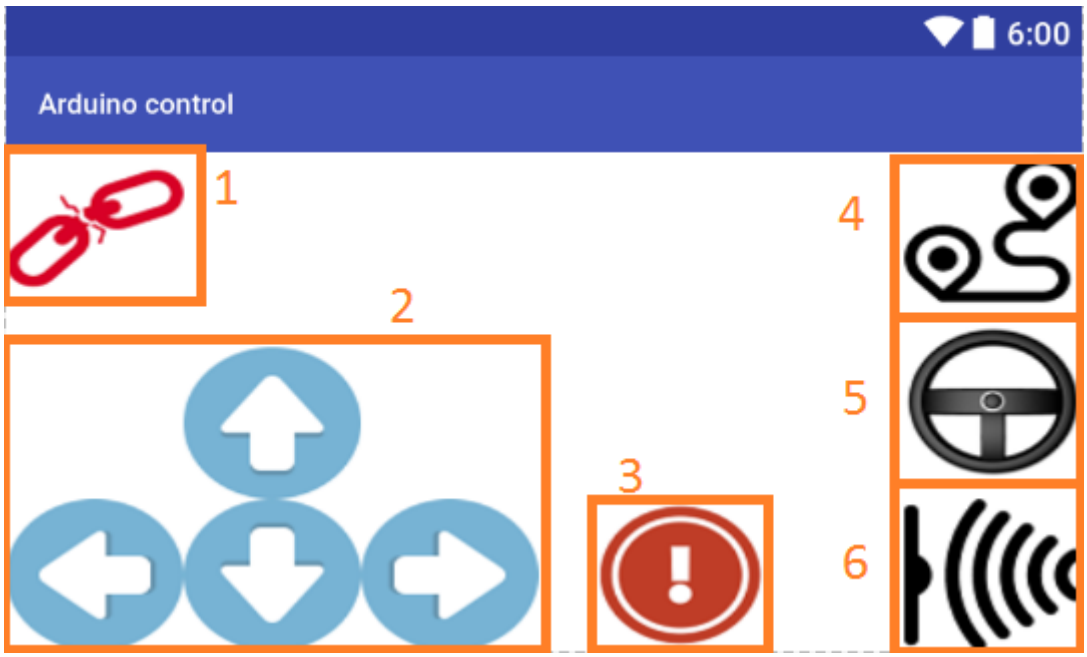


Ekran główny z listą sparowanych urządzeń:



Ekran do sterowania robotem:

1. **Rozłącz**
2. **Sterowanie ręczne za pomocą przycisków**
3. **Zatrzymanie**
4. **Wybór algorytmu jazdy po linii (czujniki wykrywające linię)**
5. **Wybór sterowania ręcznego**
6. **Wybór algorytmu jazdy przy pomocy czujników zbliżeniowych (Ultrasonic)**



Implementacja kodu Java w Android Studio

Inicjalizacja bibliotek:

```
import android.content.Intent;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.widget.TextView;
import android.widget.Toast;
import java.util.ArrayList;
import java.util.Set;
import android.widget.TextView;
import android.widget.Toast;
import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.os.AsyncTask;
import java.io.IOException;
import java.util.UUID;
```

Biblioteki są wymagane do obsługi widgetów, wątków, bluetooth, wyjątków, które mogą wystąpić

Zmienne zainicjowane oraz zainicjalizowane dla przycisków używanych do łączenia oraz kontroli nad robotem:

```
Button buttonPaired;
ListView devicesList;
buttonPaired = (Button)findViewById(R.id.button);
devicesList = (ListView)findViewById(R.id.listView);
Button btnForward,
    btnBackward,
    btnDisconnect,
    btnLeft,
    btnRight,
    btnAutomatic,
    btnSensor,
    btnManual,
    btnStop;
btnStop = (Button) findViewById(R.id.button8);
btnForward = (Button) findViewById(R.id.button2);
btnBackward = (Button) findViewById(R.id.button3);
btnDisconnect = (Button) findViewById(R.id.button4);
btnLeft = (Button) findViewById(R.id.button6);
```

```
btnRight = (Button) findViewById(R.id.button5);
btnAutomatic = (Button) findViewById(R.id.button12);
btnSensor = (Button) findViewById(R.id.button10);
btnManual = (Button) findViewById(R.id.button11);
textView = (TextView) findViewById(R.id.textView);
```

Przypisanie do zmiennej UUID:

```
//uuid Jest to globalnie unikatowy identyfikator składający się z 32 liczb szesnastkowych
static final UUID myUUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
```

Przypisanie adaptera bluetooth:

```
private BluetoothAdapter btAdapter = null;
```

Tworzenie listy sparowanych urządzeń:

```
private Set<BluetoothDevice> pairedDevices;
```

Przypisanie adresu urządzenia z aplikacją:

```
public static String myDevice_ADDRESS = "device_address";
```

Metoda sprawdzająca czy urządzenie ma modul bluetooth:

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_devices_list);

  buttonPaired = (Button)findViewById(R.id.button);
  devicesList = (ListView)findViewById(R.id.listView);

  //jesli jest modul BT
  btAdapter = BluetoothAdapter.getDefaultAdapter();

  if(btAdapter == null)
  {
     //informacja o braku modułu Bluetooth
     Toast.makeText(getApplicationContext(), "Brak modułu Bluetooth",
Toast.LENGTH_LONG).show();

     //koniec
     finish();
  }
  else if(!btAdapter.isEnabled())
  {
     //Poproś użytkownika o włączenie BlueTooth
     Intent turnBTon = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
     startActivityForResult(turnBTon,1);
  }
```

```
  buttonPaired.setOnClickListener(new View.OnClickListener() {
     @Override
     public void onClick(View v)
     {
        listOfPairedDevices();
     }
  });

}
```

Medota wypisująca liste sparowanych urządzeń:

```
private void listOfPairedDevices()
{
  pairedDevices = btAdapter.getBondedDevices();
  ArrayList list = new ArrayList();

  if (pairedDevices.size()>0)
  {
     for(BluetoothDevice bt : pairedDevices)
     {
        list.add(bt.getName() + "\n" + bt.getAddress()); //wyciaga nazwy urzadzen
     }
  }
  else
  {
     Toast.makeText(getApplicationContext(), "Brak sparowanych urządzeń.",
Toast.LENGTH_LONG).show();
  }
  final ArrayAdapter adapter = new ArrayAdapter(this,android.R.layout.simple_list_item_1,
list);
  devicesList.setAdapter(adapter);
  devicesList.setOnItemClickListener(myListClickListener);
}
```

Przycisk wykorzystujacy metodę listOfPairedDevices():

```
buttonPaired.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v)
  {
     listOfPairedDevices();
  }});
```

Metoda odpowiedzialna za możliwość kliknięcia listy urządzeń i połączenie się z nim:

```
private AdapterView.OnItemClickListener myListClickListener = new
AdapterView.OnItemClickListener()
{
```

```
  public void onItemClick (AdapterView<?> av, View v, int arg2, long arg3)
  {
     // wyciaga adres MAC urządzenia
     String info = ((TextView) v).getText().toString();
     String address = info.substring(info.length() - 17);

     Intent i = new Intent(DevicesList.this,arduinoControl.class);

     //przejscie do drugiego layout
     i.putExtra(myDevice_ADDRESS, address); //przesłanie do drugiego layout
     startActivity(i);
  }
};
```

Metoda odpowiedzialna za wyświetlanie komunikatów:

```
private void message(String s)
{
  Toast.makeText(getApplicationContext(),s,Toast.LENGTH_LONG).show();
}
```

Klasa odpowiedzialna za połączenie z drugim urządzeniem:

```
private class ConnectBT extends AsyncTask<Void, Void, Void>
{
  private boolean Connected = true;

  @Override
  protected void onPreExecute()
  {
     progress = ProgressDialog.show(arduinoControl.this, "Łączenie...", "Proszę czekać!!");
  }

  @Override
  protected Void doInBackground(Void... devices)
  {
     try
     {
        if (btSocket == null || !isBtConnected)
        {
           btAdapter = BluetoothAdapter.getDefaultAdapter();//modul bt
           BluetoothDevice dispositivo = btAdapter.getRemoteDevice(adress);//laczenie z
urzadzenie i sprawdzenie czy polaczenie jest mozliwe
           btSocket =
dispositivo.createInsecureRfcommSocketToServiceRecord(myUUID);//tworzenie RFCOMM
(SPP) polaczenia
           BluetoothAdapter.getDefaultAdapter().cancelDiscovery();
           btSocket.connect();//rozpoczecie polaczenia
        }
     }
```

```
        catch (IOException e)
        {
            Connected = false;
        }
        return null;
    }
    @Override
    protected void onPostExecute(Void result)//sprawdz
    {
        super.onPostExecute(result);

        if (!Connected)
        {
            message("Nieudane połączenie. Czy urządzenie obsługuje protokół SPP? Spróbuj
ponownie.");
            finish();
        }
        else
        {
            message("Połączono.");
            isBtConnected = true;
        }
        progress.dismiss();
    }
}
```

Metoda dodająca menu w pierwszym widoku:

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        getMenuInflater().inflate(R.menu.menu_devices_list, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

Metoda odpowiedzialna za jazdę do przodu:

```
private void goForward()
{
  if (btSocket!=null)
  {
     try
     {
        btSocket.getOutputStream().write("F".toString().getBytes());
     }
     catch (IOException e)
     {
        message("Błąd");
     }
  }
}
```

Metoda odpowiedzialna za jazdę do tyłu:

```
private void goBackward()
{
    if (btSocket!=null)
    {
        try
        {

btSocket.getOutputStream().write("B".toString().getBytes());
        }
        catch (IOException e)
        {
            message("Błąd");
        }
    }
}
```

Metoda odpowiedzialna za skręcanie w lewo:

```
private void turnLeft()
{
    if (btSocket!=null)
    {
        try
        {

btSocket.getOutputStream().write("L".toString().getBytes());
        }
        catch (IOException e)
        {
            message("Błąd");
        }
    }
}
```

Metoda odpowiedzialna za skręcanie w prawo:

```
private void turnRight()
{
  if (btSocket!=null)
  {
     try
     {
        btSocket.getOutputStream().write("R".toString().getBytes());
     }
     catch (IOException e)
     {
        message("Błąd");
     }
  }
}
```

Metoda odpowiedzialna za zatrzymanie robota:

```
private void STOP()
{
  if (btSocket!=null)
  {
     try
     {
        btSocket.getOutputStream().write("O".toString().getBytes());
     }
     catch (IOException e)
     {
        message("Błąd");
     }
  }
}
```

Metoda odpowiedzialna za wybór sterowania manualnego:

```
private void Manual()
{
  if (btSocket!=null)
  {
     try
     {
        btSocket.getOutputStream().write("M".toString().getBytes());
     }
     catch (IOException e)
     {
        message("Błąd");
     }
  }
}
```

Metoda odpowiedzialna za wybór jazdy autonomicznej z wykorzystaniem czujników zbliżeniowych:

```
private void Sensor()
{
  if (btSocket!=null)
  {
     try
     {
        btSocket.getOutputStream().write("S".toString().getBytes());
     }
     catch (IOException e)
     {
        message("Błąd");
     }
  }
}
```

Metoda odpowiedzialna za wybór jazdy autonomicznej z wykorzystaniem czujników wykrywających linię:

```
private void Automatic()
{
  if (btSocket!=null)
  {
     try
     {
        btSocket.getOutputStream().write("L".toString().getBytes());
     }
     catch (IOException e)
     {
        message("Błąd");
     }
  }
}
```

Kod przycisku odpowiedzialnego za jazdę do przodu:

```
btnForward.setOnTouchListener(new View.OnTouchListener()
{
  @Override
  public boolean onTouch(View v, MotionEvent event) {
     if(event.getAction()==MotionEvent.ACTION_DOWN)
     {
        goForward();
        return true;
     }
     if (event.getAction()==MotionEvent.ACTION_UP)
     {
```

```
            STOP();
            return true;
        }
            return false;
    }
});
```

Kod przycisku odpowiedzialnego za jazdę w tył:

```
btnBackward.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            goBackward();
            return true;
        }
        if (event.getAction() == MotionEvent.ACTION_UP) {
            STOP();
            return true;
        }
        return false;
    }
});
```

Kod przycisku odpowiedzialnego za skręcanie w lewo:

```
btnLeft.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            turnLeft();
            return true;
        }
        if (event.getAction() == MotionEvent.ACTION_UP) {
            STOP();
            return true;
        }
        return false;
    }
});
```

Kod przycisku odpowiedzialnego za skręcanie w prawo:

```
btnRight.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            turnRight();
            return true;
        }
        if (event.getAction() == MotionEvent.ACTION_UP) {
```

```
            STOP();
            return true;
        }
        return false;
    }
});
```

Kod przycisku odpowiedzialnego za zatrzymanie robota:

```
    btnStop.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            STOP(); //stop
        }
    });
}
```

Kod przycisku odpowiedzialnego za wybór sterowania manualnego:

```
 btnManual.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Manual(); //reczne
    }
});
```

Kod przycisku odpowiedzialnego za wybór sterowania autonomicznego z wykorzystaniem czujników wykrywających linię:

```
btnAutomatic.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Automatic(); //
    }
});
```

Kod przycisku odpowiedzialnego za wybór jazdy autonomicznej z wykorzystaniem czujników zbliżeniowych:

```
btnSensor.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Sensor(); //sensor
    }
});
```

Kod metody odpowiedzialny za wyświetlanie okien dialogowych:

```
private void message(String s)
{
    Toast.makeText(getApplicationContext(),s,Toast.LENGTH_LONG).show();
}
```

Kod odpowiedzialny za implementację w widokumenu:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
   getMenuInflater().inflate(R.menu.menu_arduino_control, menu);
   return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
   int id = item.getItemId();

   if (id == R.id.action_settings) {
      return true;
   }

   return super.onOptionsItemSelected(item);
}
```

System przesyłający video streaming pomiędzy telefonami z systemem Android.

OPIS

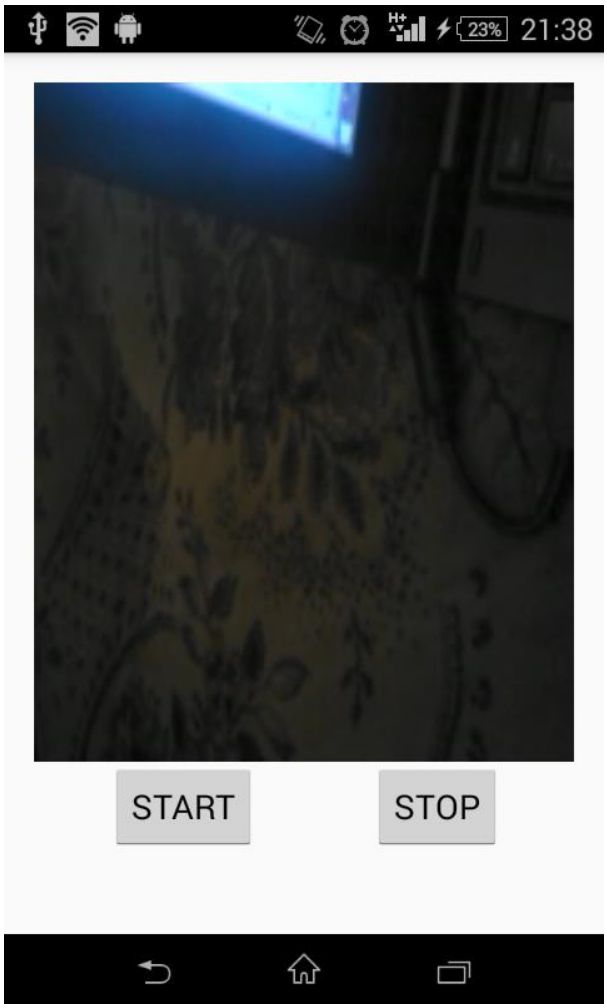
W skład systemu do przesyłania video streamingu wchodzą dwie aplikacje. Jedną z nich jest aplikacja Serwera. Jest to program zainstalowany na telefonie z systemem Android, zamontowanym na robocie. Druga aplikacja jest to program Kliencki, który odbiera przesyłane dane w postaci video streamingu. Jest to również aplikacja działająca w systemie Android.

Obie aplikacje napisane zostały ze zgodnością Android API od wersji 15. Oznacza to, że wszystkie telefony z systemem Android w wersji co najmniej 4.0.3 (IceCreamSandwich) będą obsługiwały niniejszą aplikację.

Aplikacja Serwera łączy się w sieci lokalnej z Internetem przez Wi-Fi oraz otwiera zasoby natywne telefonu, tj. otwiera moduł kamery. Aplikacja Kliencka korzystając z osadzonej przeglądarki internetowej łączy się z przydzielonym przez Serwer adresem IP w sieci lokalnej i tym samym jest w stanie odtworzyć nadawany pakiet z video streamingiem.

Przy tworzeniu rzeczonych aplikacji użyto narzędzi Android Studio oraz środowiska Java JDK w wersji 1.8.0_77.

APLIKACJA SERWERA



Aplikacja ma za zadanie połączyć się z natywnym modułem kamery oraz wydaniem prośby akceptacji połączenia z danym IP oraz przyjęcia pakietów danych w lokalnej sieci. Dzieje się to po naciśnięciu przycisku START. Tuż przed tym jednak użytkownik musi manualnie w telefonie nawiązać połączenie z siecią lokalną poprzez Wi-Fi bądź w przypadku korzystania z pakietu internetowego przez operatora telefonii komórkowej włączyć HotSpot (aktywować usługę tetheringu) i nadawać sieć bezprzewodową udostępnioną również innym urządzeniom w tym urządzeniu z aplikacji Klienckiej.

Struktura aplikacji Serwera w formacie XML jest następująca:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.mar.stream2.VideoServer">


    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="START"
        android:id="@+id/btn_start"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginLeft="40dp"
        android:layout_marginStart="40dp"
        android:layout_marginBottom="28dp" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="STOP"
        android:id="@+id/btn_stop"
        android:layout_alignBottom="@+id/btn_start"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:layout_marginRight="38dp"
        android:layout_marginEnd="38dp" />

    <SurfaceView
        android:layout_width="match_parent"
```

```
        android:layout_height="match_parent"
        android:id="@+id/surfaceView1"
        android:layout_above="@+id/btn_start"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />
</RelativeLayout>
```

Na wyżej zadeklarowane w formacie XML pojemniki oddziałują funkcje Java w postaci:

```
import android.app.Activity;
import android.hardware.Camera;
import android.os.Bundle;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class VideoServer extends Activity implements
SurfaceHolder.Callback {
   TextView testView;

   Camera camera;
   SurfaceView surfaceView;
   SurfaceHolder surfaceHolder;

   private final String tag = "VideoServer";

   Button start, stop;
   int rotation;

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_video_server);

       start = (Button)findViewById(R.id.btn_start);
       start.setOnClickListener(new Button.OnClickListener() {
           public void onClick(View arg0) {
               start_camera();
           }
       });

       stop = (Button)findViewById(R.id.btn_stop);
```

```
        stop.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View arg0) {
                stop_camera();
            }
        });

        surfaceView = (SurfaceView)findViewById(R.id.surfaceView1);
        surfaceHolder = surfaceView.getHolder();
        surfaceHolder.addCallback(this);

surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    private void start_camera() {
        camera = null;
        try{
            camera = Camera.open();
        }catch(RuntimeException e){
            Log.e(tag, "init_camera: " + e);
            return;
        }
        Camera.Parameters param;
        param = camera.getParameters();
        rotation =
this.getWindowManager().getDefaultDisplay().getRotation();
        Toast.makeText(this, "rotation = " + rotation,
Toast.LENGTH_LONG).show();

        //modify parameter
        param.setPreviewFrameRate(20);
        param.setPreviewSize(176, 144);
        param.setRotation(90);
        camera.setParameters(param);
        try {
            camera.setPreviewDisplay(surfaceHolder);
            camera.startPreview();
        } catch (Exception e) {
            Log.e(tag, "init_camera: " + e);
            return;
        }
    }

    private void stop_camera() {
        camera.stopPreview();
        camera.release();
    }
}
```

Na podstawie powyższego kodu w funkcji onCreate() odnajdujemy przyciski zadeklarowane w XML (przyciski START i STOP) oraz przypisujemy im funkcję onClickListenera, żeby przyciski nasłuchiwały, gdy użytkownik zdecyduje się nacisnąć któryś z nich. Ponadto, w funkcji inicjowany jest widget surfaceView, do którego to będzie wprowadzany podgląd video z obecnego w telefonie moduły kamery.

Naciskając przycisk START użytkownik nawiązuje połączenie z kamerą w telefonie oraz wysyła żądanie akceptacji połączenia innego urządzenia ze sobą poprzez wyznaczony adres IP.

Aplikacja Kliencka po połączeniu z lokalną sieci, w której znajduje się urządzenie Serwera otrzymuje w widgecie okna przeglądarki podany adres, który wpisuje w pasku adresów http.

Implementacja Fuzzy Logic w Arduino

Wymagane biblioteki:

- **eFFL (Embedded Fuzzy Logic Library) - https://github.com/zerokol/eFLL**

Inicjalizacja bilbiotek:

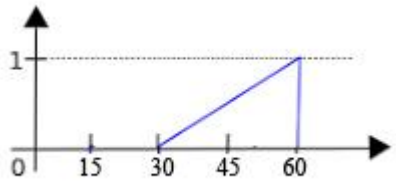
```
#include <Fuzzy.h>
#include <FuzzyComposition.h>
#include <FuzzyInput.h>
#include <FuzzyIO.h>
#include <FuzzyOutput.h>
#include <FuzzyRule.h>
#include <FuzzyRuleAntecedent.h>
#include <FuzzyRuleConsequent.h>
#include <FuzzySet.h>
```

Zasady działania zostały napisane w C/C++, wykorzystują standardową bibliotekę stdlib.h. Nie posiadają żadnych ograniczeń na ilość zbiorów/zasad, jedynym limitem jest zasobność i szybkość mikrokontrolera.

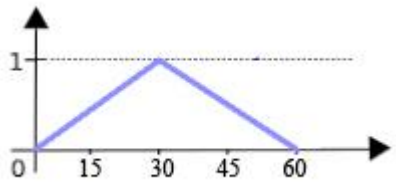
Używane obiekty:

FuzzySet - obiekt zawierający definicję jednego termu należącego do zmiennej lingwistycznej. Jako parametry podajemy wartości dla 4 zmiennych typu float (A,B,C,D) na podstawie których biblioteka odczytuje rodzaj zbioru (singletonowy, trójkątny itp.)(np. duża odległość)

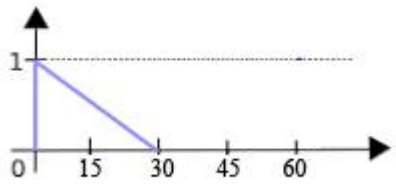
FuzzySet* big = new FuzzySet(30,60,60,60);



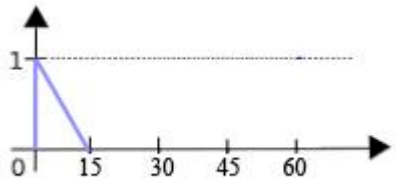
FuzzySet* average = new FuzzySet(0,30,30,60);



FuzzySet* small = new FuzzySet(0,0,0,30);



FuzzySet* very_small = new FuzzySet(0,0,0,15);



FuzzyInput - Grupuje wszystkie zbiory rozmyte należące do określonej domeny (np. odległość)

Inicjalizacja (gdzie 1 to ID do którego odwołujemy się przy procesie fuzzyfikacji):

FuzzyInput* distance = new FuzzyInput(1);

Dodanie wcześniej zdefiniowanych zbiorów:

```
distance->addFuzzySet(big);
distance->addFuzzySet(average);
distance->addFuzzySet(small);
distance->addFuzzySet(very_small);
```

Dodanie nowo utworzonego wejścia do sterownika rozmytego:

```
fuzzy->addFuzzyInput(distance);
```

Wykorzystanie FuzzyInput w projekcie przy odczytu odległosci z 3 sensorów:

```
// INPUT
// odleglosc na sensorze srodkowym
FuzzyInput* distance = new FuzzyInput(1);
//
FuzzySet* big = new FuzzySet(30,60,60,60);
distance->addFuzzySet(big);
FuzzySet* average = new FuzzySet(0,30,30,60);
distance->addFuzzySet(average);
FuzzySet* small = new FuzzySet(0,0,0,30);
FuzzySet* very_small = new FuzzySet(0,0,0,15);
distance->addFuzzySet(very_small);
//
fuzzy->addFuzzyInput(distance);
// odleglosc na sensorze lewym
FuzzyInput* distance2 = new FuzzyInput(2);
//
FuzzySet* big2 = new FuzzySet(30,60,60,60);
distance2->addFuzzySet(big2);
FuzzySet* average2 = new FuzzySet(0,30,30,60);
distance2->addFuzzySet(average2);
FuzzySet* small2 = new FuzzySet(0,0,0,30);
distance2->addFuzzySet(small2);
FuzzySet* very_small2 = new FuzzySet(0,0,0,15);
distance2->addFuzzySet(very_small2);
//
fuzzy->addFuzzyInput(distance2);
// odleglosc na sensorze prawym
FuzzyInput* distance3 = new FuzzyInput(3);
//
FuzzySet* big3 = new FuzzySet(30,60,60,60);
distance3->addFuzzySet(big3);
FuzzySet* average3 = new FuzzySet(0,30,30,60);
distance3->addFuzzySet(average3);
FuzzySet* small3 = new FuzzySet(0,0,0,30);
distance3->addFuzzySet(small3);
FuzzySet* very_small3 = new FuzzySet(0,0,0,15);
distance2->addFuzzySet(very_small3);
//
fuzzy->addFuzzyInput(distance3);
```

FuzzyOutput - grupuje wszystkie zbiory należące do określonej domeny związanej z wartością wyjściową.

Inicjalizacja (gdzie 1 to ID do którego odwołujemy się przy procesie defuzzyfikacji):

```
FuzzyOutput* speed = new FuzzyOutput(1);
```

Tworzenie odpowiedniego FuzzySet:

```
FuzzySet* slow = new FuzzySet(0,150,150,150);
FuzzySet* mild = new FuzzySet(0,150,150,255);
FuzzySet* fast = new FuzzySet(150,255,255,255);
```

Dodanie FuzzySets do FuzzyOutput:

```
speed->addFuzzySet(slow);
speed->addFuzzySet(mild);
speed->addFuzzySet(fast);
```

Dodanie nowo utworzonego wyjścia do sterownika rozmytego:

```
fuzzy->addFuzzyOutput(speed);
```

Wykorzystanie FuzzyOutput w projekcie do obliczania prędkości (output), wartości zwrotu skrętu (output2), kierunku skrętu (output3)

```
// OUTPUT

 // predkosc
 FuzzyOutput* speed = new FuzzyOutput(1);
 //
 FuzzySet* slow = new FuzzySet(0, 150, 150, 150);
 speed->addFuzzySet(slow);
 //
 FuzzySet* mild = new FuzzySet(0, 150, 150, 255);
 speed->addFuzzySet(mild);
 //
 FuzzySet* fast = new FuzzySet(150, 255, 255, 255);
 speed->addFuzzySet(fast);
 //
 fuzzy->addFuzzyOutput(speed);

 // zakręt
 FuzzyOutput* turn = new FuzzyOutput(2);
```

```
//
FuzzySet* small_turn = new FuzzySet(150, 150, 150, 450);
turn->addFuzzySet(small_turn);
//
FuzzySet* mild_turn = new FuzzySet(150, 450, 450, 900);
turn->addFuzzySet(mild_turn);
//
FuzzySet* fast_turn = new FuzzySet(450, 900, 900, 900);
turn->addFuzzySet(fast_turn);
//
fuzzy->addFuzzyOutput(turn);

// kierunek (lewo, prawo, nieokreślony)
FuzzyOutput* side = new FuzzyOutput(3);
//
FuzzySet* left = new FuzzySet(0, 0, 0, 5);
side->addFuzzySet(left);
//
FuzzySet* right = new FuzzySet(5, 10, 10, 10);
side->addFuzzySet(right);
//
FuzzySet* unknown = new FuzzySet(10, 15, 15, 15);
side->addFuzzySet(unknown);
//
fuzzy->addFuzzyOutput(side);
```

FuzzyRuleAntecedent - obiekt zawierający wymagania poprzedzające FuzzyRule, jako argumenty podajemy konkretne FuzzySets które możemy łączyć za pomocą kwantyfikatorów OR/AND (np. duży dystans na przednim sensorze i mały na lewym). Funkcje przypisujące zbiory rozmyte mogą być dwu lub jedno argumentowe, jedna jako argument możemy podać także inny FuzzyRuleAntecedent by tworzyć bardziej złożone poprzedniki.

Inicjalizacja:

```
// (jeśli Lewo i Prawo Duże, Jedź do przodu szybko)
FuzzyRuleAntecedent* ifLeftAndRightBig = new FuzzyRuleAntecedent();
FuzzyRuleAntecedent* ifLeftAndForwardAndRightBig = new FuzzyRuleAntecedent();
```

Połączenie odpowiednią metodą z odpowiednimi termami:

```
ifLeftAndRightBig->joinWithAND(big2,big3);
ifLeftAndForwardAndRightBig->joinWithAND(ifLeftAndRightBig,big);
```

FuzzyRuleConsequente - zasada działania podobna jak FuzzyRuleAntecedent, jednak tym razem opisujemy następnik FuzzyRule. Po utworzeniu nowego obiektu zamiast łączyć FuzzySets w pary, dopisujemy do niego nieograniczoną ilość pojednyńczych zbiorów (np. mocny skręt, wysoka prędkość)

Inicjalizacja:

```
FuzzyRuleConsequent* GoForwardFast = new FuzzyRuleConsequent();
```

Dodanie pojedyńczych termów:

```
GoForwardFast->addOutput(fast);
```

FuzzyRule - Podstawowy obiekt zarządzający logiką. Budowa : (ID, antecedent, consequent) gdzie ID to identyfikator dla FuzzyRule (przydatny w sprawdzaniu która zasada została aktywowana), Antecedent i Consequent to poprzednik i następnik (np. 1, jeśli wszystkie sensor mają dużo miejsca, to jedź szybko)

Inicjalizacja:

```
FuzzyRule* fuzzyRule01 = new FuzzyRule(1,ifLeftAndForwardAndRightBig,GoForwardFast);
```

Dodanie do sterownika rozmytego:

```
fuzzy->addFuzzyRule(fuzzyRule01);
```

Wykorzystanie FuzzyRules do jazdy w projekcie:

```
// ZASADY LOGIKI ROZMYTEJ

// Pierwsza zasada (jesli Lewo i Prawo Duze, Jedz do przodu szybko)
FuzzyRuleAntecedent* ifLeftAndRightBig = new FuzzyRuleAntecedent();
ifLeftAndRightBig->joinWithAND(big2, big3);

FuzzyRuleAntecedent* ifLeftAndForwardAndRightBig = new FuzzyRuleAntecedent();
ifLeftAndForwardAndRightBig->joinWithAND(ifLeftAndRightBig, big);

FuzzyRuleConsequent* GoForwardFast = new FuzzyRuleConsequent();
GoForwardFast->addOutput(fast);

FuzzyRule* fuzzyRule01 = new FuzzyRule(1, ifLeftAndForwardAndRightBig, GoForwardFast);
fuzzy->addFuzzyRule(fuzzyRule01);
```

```
// Druga zasada (jeśli lewo bardzo małe skręć w prawo)
  FuzzyRuleConsequent* DoARightSmallTurn = new FuzzyRuleConsequent();
  DoARightSmallTurn->addOutput(small_turn);
  DoARightSmallTurn->addOutput(right);

  FuzzyRuleAntecedent* ifLeftVerySmall = new FuzzyRuleAntecedent();
  ifLeftVerySmall->joinSingle(very_small2);

  FuzzyRule* fuzzyRule02 = new FuzzyRule(2, ifLeftVerySmall, DoARightSmallTurn);
  fuzzy->addFuzzyRule(fuzzyRule02);

  // Trzecia zasada (jeśli prawo bardzo małe skręć w lewo)
  FuzzyRuleAntecedent* ifRightVerySmall = new FuzzyRuleAntecedent();
  ifRightVerySmall->joinSingle(very_small3);

  FuzzyRuleConsequent* DoALeftSmallTurn = new FuzzyRuleConsequent();
  DoALeftSmallTurn->addOutput(small_turn);
  DoALeftSmallTurn->addOutput(left);

  FuzzyRule* fuzzyRule03 = new FuzzyRule(3, ifRightVerySmall, DoALeftSmallTurn);
  fuzzy->addFuzzyRule(fuzzyRule03);

  // Czwarta zasada (main) (jeśli lewo średnie lub duże i prawo średnie lub duże i przód średni jedź z
średnią prędkością)
  FuzzyRuleAntecedent* ifLeftAverageOrBig = new FuzzyRuleAntecedent();
  ifLeftAverageOrBig->joinWithOR(average2, big2);

  FuzzyRuleAntecedent* ifRightAverageOrBig = new FuzzyRuleAntecedent();
  ifRightAverageOrBig->joinWithOR(average3, big3);

  FuzzyRuleAntecedent* ifLeftAverageOrBigAndRightAverageOrBig = new
FuzzyRuleAntecedent();
  ifLeftAverageOrBigAndRightAverageOrBig->joinWithAND(ifLeftAverageOrBig,
ifRightAverageOrBig);

  FuzzyRuleAntecedent* ifLeftAverageOrBigAndRightAverageOrBigAndForwardAverage = new
FuzzyRuleAntecedent();
  ifLeftAverageOrBigAndRightAverageOrBigAndForwardAverage-
>joinWithAND(ifLeftAverageOrBigAndRightAverageOrBig, average);

  FuzzyRuleConsequent* GoForwardMild = new FuzzyRuleConsequent();
  GoForwardMild->addOutput(mild);

  FuzzyRule* fuzzyRule04 = new FuzzyRule(4,
ifLeftAverageOrBigAndRightAverageOrBigAndForwardAverage, GoForwardMild);
  fuzzy->addFuzzyRule(fuzzyRule04);
```

```
// Zasada piąta (jeśli przedni sensor podaje małe odległości)
  FuzzyRuleConsequent* DoAMildTurn = new FuzzyRuleConsequent();
  DoAMildTurn->addOutput(mild_turn);
  DoAMildTurn->addOutput(unknown);

  FuzzyRuleAntecedent* ifForwardVerySmall = new FuzzyRuleAntecedent();
  ifForwardVerySmall->joinSingle(very_small);

  FuzzyRule* fuzzyRule05 = new FuzzyRule(5, ifForwardVerySmall, DoAMildTurn);
  fuzzy->addFuzzyRule(fuzzyRule05);


  //Zasada szósta (jeśli lewo średnie lub prawo średnie i przód duży, jedź z średnią prędkością)
  FuzzyRuleAntecedent* ifLeftAverageAndRightAverage = new FuzzyRuleAntecedent();
  ifLeftAverageAndRightAverage->joinWithOR(average2, average3);

  FuzzyRuleAntecedent* ifLeftAverageAndRightAverageAndForwardBig = new
FuzzyRuleAntecedent();
  ifLeftAverageAndRightAverageAndForwardBig->joinWithAND(ifLeftAverageAndRightAverage,
big);

  FuzzyRule* fuzzyRule06 = new FuzzyRule(6, ifLeftAverageAndRightAverageAndForwardBig,
GoForwardMild);
  fuzzy->addFuzzyRule(fuzzyRule06);
```

Sensory ultradźwiękowe wykorzystane w robocie mają swoje ograniczenia - zarówno przy dalszych odległościach jak przy ich całkowitym zakryciu zwracają wartość 0. Dla prawidłowego funkcjonowania obecnie użytych zbiorów rozmytych nie powinny zwracać również wartości powyżej 60. Dlatego wartość z sensorów wprowadzamy do dodatkowej funkcji limit.

```
int limit(int value, int sensor)
{
   if ( value > 60)
  {
        value = 60;
  }
   if ( value == 0)
  {
        value = 60;
  }
  return value;
}
```

Proces fuzzify/defuzzify - mając sterownik wraz z zasadami przypisujemy konkretne wartości pod FuzzyInput, następnie wyciągamy wartości z FuzzyOutput

Inicjalizacja input:

```
fuzzy->setInput(1, limit(middleEye.ping_cm(),1)); // środkowy sensor
fuzzy->setInput(2, limit(leftEye.ping_cm(),2)); // lewy sensor
fuzzy->setInput(3, limit(rightEye.ping_cm(),3)); // prawy sensor
```

Włączenie sterownika:

```
fuzzy->fuzzify();
```

Inicjalizacja output:

```
int output = fuzzy->defuzzify(1); // prędkość
int output2 = fuzzy->defuzzify(2); // wartość skrętu
int output3 = fuzzy->defuzzify(3); // kierunek skrętu
```

Posiadając wartości wyliczone przez sterownik rozmyty, możemy wykorzystać je w funkcjach sterujących robotem.

Podstawowe zasaday działania algorytmu :

- Wartość skrętu jest obliczana tylko kiedy czujniki zwracają przeszkodę na drodzę, dlatego gdy wartość output2 wynosi 0 robot jedzie do przodu.
- Kiedy wartość output2 jest różna od 0, na podstawie output3 (zakres od 1 do 5 , lub od 6 do 9, lub od 10 do 15) wybierany jest kierunek skrętu
- Kiedy przedni czujnik zwraca bardzo małą wartość, output3 jest wyliczany w przedziale od 10 do 15 . Wtedy w zależności od dyskretnej wartości czujników wykonujemy skręt w odpowiednią stronę. Jest to próba optymalizacji systemu, podobne zachowania można byłoby dopisać korzystając z większej ilości zbiorów rozmytych. W przypadku naszego sprzętu ogranicza nas jego zasobność i moc obliczeniowa.
- Jeśli zaistnieje sytuacja w której robot ma problemy z wyjechaniem np. z rogu i zaczyna skręcać naprzemiennie w lewo i w prawo, po sekwencji 3 skrętów odjeżdża do tyłu i na podstawie dyskretnej wartości czujników wykonuje mocny skręt w określonym kierunku. W tym celu korzystamy z pomocniczych zmiennych left i right.

Algorytm:

```
if ( output3 >= 5 && output3 < 10)
 {
        Serial.println("Prawo o mocy:");
        Serial.println(output2);
        motorsTurnRight(output2, 125);
        if (left == 1)
        {
                if (right == 0)
                {
                        right++;
                        left = 2;
                }
        }
 }
 else
 {
        if ( output3 > 0 && output3 < 10)
        {
                Serial.println("Lewo o mocy:");
                Serial.println(output2);
                motorsTurnLeft(output2, 125);
                if (left == 0)
                {
                        left++;
                }
                if (left == 2)
                {
                        motorsBackward(225, 125);
                        left = 0;
                        right = 0;
                        if ( limit(leftEye.ping_cm(), 2) >= limit(rightEye.ping_cm(), 3))
                        {
                                motorsTurnLeft(800, 125);
                        }
                        else
                        {
                                motorsTurnRight(800, 125);
                        }
                }
        }
        else
        {
                if ( output3 > 0 )
                {
                        if ( limit(leftEye.ping_cm(), 2) >= limit(rightEye.ping_cm(), 3))
```

```
                {
                        motorsTurnLeft(output2, 125);
                        Serial.println("Inne: Lewo o mocy:");
                        Serial.println(output2);

                        if (left == 0)
                        {
                                left++;
                        }

                        if (left == 2)
                        {
                        motorsBackward(225, 125);
                        left == 0;
                        right == 0;
                        if ( limit(leftEye.ping_cm(), 2) >= limit(rightEye.ping_cm(), 3))
                        {
                        motorsTurnLeft(800, 125);
                        }
                        else
                        {
                                motorsTurnRight(800, 125);
                        }
                }

        }
        else
        {
                motorsTurnRight(output2, 125);
                Serial.println("Inne :Prawo o mocy:");
                Serial.println(output2);
                if (left == 1)
                {
                        if (right == 0)
                        {
                        right++;
                        left == 2;
                        }
                }


        }
}
else
{
        if (output > 170)
        {
        motorsForward(40, output);
```

```
                left = 0;
                right = 0;
                }
                else
                {
                motorsForward(25, output);
                left = 0;
                right = 0;
                }
        }
    }
}
```

Funkcja Serial.println jest wykorzystywana do wypisywania wartości ze sterownika oraz sensorów podczas debugowania w Arduino IDE. Żeby sprawdzić które zasady zostały użyte przy obliczaniu wartości możemy korzystać z wbudowanej w sterowniki funkcji isFiredRule(ID);

Inicjalizacja:

```
bool wasTheRulleFired = fuzzy->isFiredRule(1);
```

W przypadku naszego projektu sprawdzenie które z naszych 6 zasad zostały uruchomione odbywa się w następujący sposób:

```
bool wasTheRulleFired = fuzzy->isFiredRule(1);
Serial.println(wasTheRulleFired);
wasTheRulleFired = fuzzy->isFiredRule(2);
Serial.println(wasTheRulleFired);
wasTheRulleFired = fuzzy->isFiredRule(3);
Serial.println(wasTheRulleFired);
wasTheRulleFired = fuzzy->isFiredRule(4);
Serial.println(wasTheRulleFired);
wasTheRulleFired = fuzzy->isFiredRule(5);
Serial.println(wasTheRulleFired);
wasTheRulleFired = fuzzy->isFiredRule(6);
Serial.println(wasTheRulleFired);
```

Przykładowy output sterownika:

```
Deffuzify
220
0
0
0
oryginalny print
60
60
60
Zasady:
1
0
0
0
0
0
Deffuzify
220
0
0
0
```

Wartość lewego, środkowego i prawego czujnika

Które zasady zostały aktywowane

Obliczona wartość prędkości, skrętu oraz wypisanie wartości left i right.

Elektronika i części mechaniczne

Części użyte przy budowie projektu:

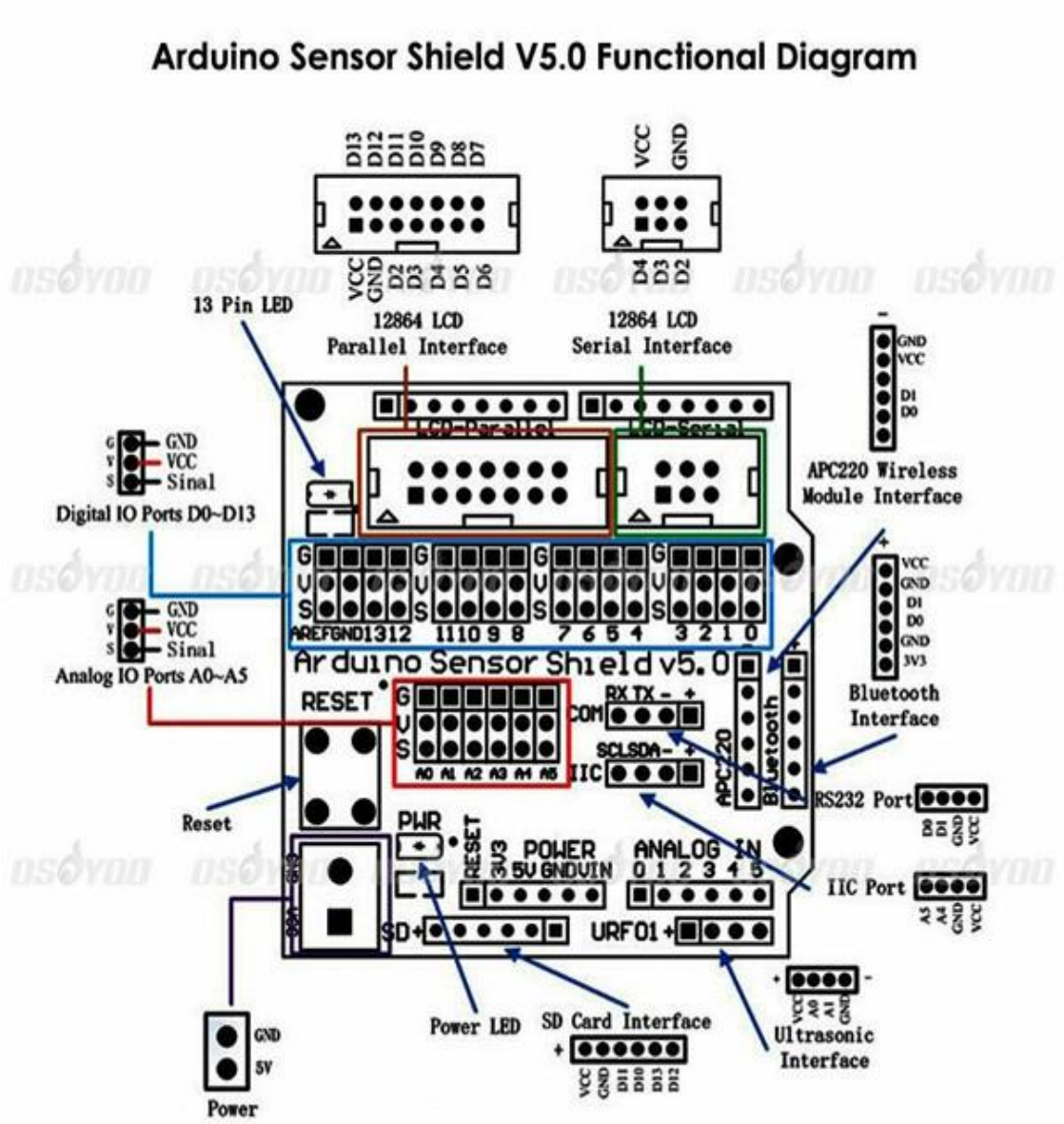
1. Szkielet z tworzywa sztucznego wraz ze wspornikami, śrubami mocującymi i montażem pod silniki.

2. Cztery silniki prądu stałego o następujących parametrach:

Working Voltage	DC 3V	DC 5V	DC 6V
Working Current	100ma	100ma	120ma
Reduction Ratio	48:1		
No-load(with wheel)	100 RPM	190 RPM	240 RPM
Wheel Diameter	6.6cm		
Speed(no-load)	20m/min	39m/min	48m/min
Weight	50g		
Size	70mm*22mm*18mm		
Noise	<65db		

3. Cztery koła wraz z oponami wykonane z tworzywa sztucznego.
4. Nadajnik-odbiornik bluetooth HC-06 z zasięgiem do 10 metrów.

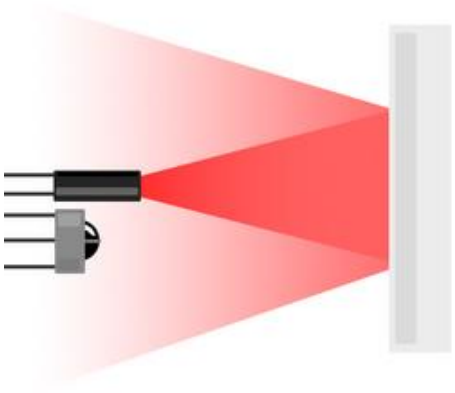
5. Nakładka rozszerzająca na Arduino UNO R3 Sensor Shield V5.



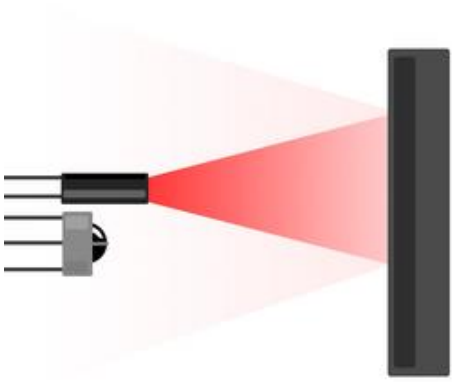
6. Płytka Arduino DCCduino ATMEGA328 oparta na Arduino UNO.
7. Koszyczek na dwie baterie 18650.
8. Dwie baterie li-po 18650 2100 mAh marki Sony.
9. Trzy czujniki podczerwieni line-tracking.
10. Trzy czujniki naddźwiękowe wykorzystywane do algorytmów zbliżeniowych.
11. Zestaw trzypinowych kabli i złączek do połączenia Arduino z czujnikami.
12. Przewody miedziane służące do połączenia zasilania do komponentów robota.

Zasada działania czujników na podczerwień:

1. Wbudowany LED wysyła światło o określonej długości fali, tej samej którą wbudowany sensor światła próbuje wykryć.
2. Światło, jeśli natrafi na jasną powierzchnię, zostaje odbite i wykryte przez sensor. W przypadku ciemnych powierzchni wysyłany sygnał zostaje przytłumiony.
3. Umożliwia robotowi wykrywanie konturu czarnych linii na białym tle, pozwalając mu na samodzielną jazdę



Jasne obiekty odbijają więcej promienia IR



Jasne obiekty odbijają mniej promienia IR

Zasada działania czujników zbliżeniowych:

1. Na przodzie czujnika znajduje się para tzw. “oczu”, jedno (trigger) wysyła sygnał ultradźwiękowy a drugie (echo) je odbiera.
2. Sygnał podróżuje do przodu dopóki nie natrafi na przeszkodę, następnie “odbija” się od niej i zostaje pobrany przez drugi czujnik.
3. Odległość do przeszkody zostaje obliczona na podstawie interwału czasowego między wysłaniem a otrzymaniem sygnału.
4. Wykrywa przeszkody na drodze. Zamontowany z przodu i po bokach robota

Instrukcja złożenia robota:

Poniższa instrukcja przedstawia krok po kroku proces złożenia robota. Pozwala również prześledzić i zrozumieć ewentualne usterki mechaniczne, mogące powstać podczas pracy z urządzeniem.

1. Przylutuj kable do silniczków. Zadbaj o ich odpowiednią długość.
2. Podłącz silniczki do odpowiednich wejść w sterowniku. Motor A to strona lewa, Motor B to silniczki po prawej stronie.
3. Pod koniec sprawdź, czy wszystkie silniki kręcą się w dobrą stronę. Jeśli nie, odwróc polaryzację przy sterowniku.
4. Zamontuj pierwszą podporę silnika do obudowy.
5. Z drugiej strony przyłóż kolejną podpórkę.
6. Przeprowadź najdłuższą śrubkę przez podpórki i silniczek, zakręć z drugiej strony nakrętkę.
7. Przymocuj do podstawy sterownik, od dołu zakręcając nakrętkę.
8. Podłącz koszyczek na baterie. Czarny kabel do GND, czerwony do VMS.

9. Podłącz do sterownika kable sygnałowe. Podłącz pod odpowiednie porty:
   enA = 9
   in1 = 10
   in2 = 11
   enB = 5
   in3 = 6
   in4 = 7
10. Przykręć górną platformę robotą.
11. Trzy czujniki line-tracking - znajdź dla nich miejsce z przodu robota. Uważaj ze zbyt mocnym dokręceniem śrubki - możesz je połamać. Podłącz czujniki do następujących portów:
    lewy = 14
    prawy = 15
    środkowy = 16
12. Wyprowadź koszyczek na górę. Może to zająć parę prób. Przykręć go śrubkami w dwóch miejscach.
13. Dolutuj do koszyczka kable, którymi zasilimy Arduino.
14. Przygotuj kabel 4-pin do podłączenia czujników zbliżeniowych. Jeśli nie posiadasz takiego możesz użyć dwóch kabli 3-pinowych odłączając od każdej ze złączek jeden kabelek, jak na zdjęciach.
15. Podłącz czujniki zbliżeniowe. Uważaj na odpowiednie podpięcie ich pod podane porty:
    middleEyeTrigger = 8
    middleEyeEcho = 4
    leftEyeTrigger = 3
    leftEyeEcho = 2
    rightEyeEcho = 12
    rightEyeTrigger = 13
    turnTest = 0
16. Zamontuj czujniki na robocie. Możesz np.: lekko wygiąć wtyki i przykleić je do szkieletu za pomocą taśmy klejącej.

Kod Arduino

Wykorzystane biblioteki:

1. eFLL (Embedded Fuzzy Logic Library) - wykorzystywana do zbiorów rozmytych. Powiązane importy:

   #include <Fuzzy.h>
   #include <FuzzyComposition.h>
   #include <FuzzyInput.h>
   #include <FuzzyIO.h>
   #include <FuzzyOutput.h>
   #include <FuzzyRule.h>
   #include <FuzzyRuleAntecedent.h>
   #include <FuzzyRuleConsequent.h>
   #include <FuzzySet.h>

2. NewPing - wykorzystywana przy obsłudze czujników zbliżeniowych. Powiązane importy:

   #include <NewPing.h>

3. SoftwareSerial - wykorzystywana przy bluetooth. Powiązane importy:

   #include <SoftwareSerial.h>

Definicje zmiennych i stałych:
W projekcie zdefiniowano następujące zmienne dotyczące portów:

1. *// motor one*
   int enA = 9;
   int in1 = 10;
   int in2 = 11;
   // motor two
   int enB = 5;
   int in3 = 6;
   int in4 = 7;

   Przypisanie portów ze sterownika silników. Motor A obsługuje silniki po lewej stronie, Motor B po prawej. EnA i enB pozwala na aktywację silników, in1-in4 na odpowiednie nimi sterowanie.

2. *// IR sensors*
   int leftIR = 15;
   int rightIR = 14;
   int centerIR = 16;

   Porty odpowiadające za kolejne czujniki podczerwieni odpowiedzialne za line-tracking.

3. *int middleEyeTrigger = 8;*
   int middleEyeEcho = 4;
   int leftEyeTrigger = 3;
   int leftEyeEcho = 2;
   int rightEyeEcho = 12;
   int rightEyeTrigger = 13;

   Porty odpowiadające za kolejne czujniki zbliżeniowe odpowiedzialne za tryb jazdy autonomicznej.

4. *int turnTest = 0;*
   int right;
   int left;

   Zmienne wykorzystywane w algorytmie ultrasonic.

5. *char bt_signal;*
   char btx;

   Zmienne globalne do których przypisywany zostaje sygnał odbierany przez nadajnik bluetooth.

6. *NewPing leftEye(leftEyeTrigger, leftEyeEcho, 100);*
   NewPing middleEye(middleEyeTrigger, middleEyeEcho, 100);
   NewPing rightEye(rightEyeTrigger, rightEyeEcho, 100);

   Obiekty biblioteki NewPing odpowiedzialne za obsługę czujników zbliżeniowych. Zastosowana biblioteka jest bardziej dokładna niż własne, podstawowe algorytmy i zapewnia większą wydajność obliczeniową.

Funkcje:

1. **Setup**

```
void setup() {
  // FUNCTION CALLED ON RESET

  // set all the control pins to outputs
  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);

  // set serial
  //Serial.begin(9600); // uncomment that for Serial to work

  // set bluetooth serial
  BT.begin(9600);       // uncomment that for BT to work
  BT.println("Terminator 0.7 connected.");

  fuzzy_rules();

}
```

Funkcja setup wywoływana jest przy każdym uruchomieniu/restarcie Arduino. Na początku przypisujemy wyjście dla poszczególnych pinów.
Następnie ustawiamy odpowiedni serial - w zależności od tego, czy chcemy nasłuchiwać sygnałów bluetooth (**BT.begin**) czy wiadomości debugowania (**Serial.begin**).
Na samym końcu wywołujemy funkcję (**fuzzy_rules**) definiującą zbiory rozmyte.

2. **listen_bluetooth**

```
void listen_bluetooth() {
  if (BT.available()) {
        bt_signal = (BT.read());
        if (bt_signal == 'O') {
              motorsOff();
        }
        while (bt_signal == 'F') {
        // drive forward
        btx = (BT.read());
        motorsForward(0, 250);
        if (btx == 'O') {
```

```
            motorsOff();
            break;
}
}
while (bt_signal == 'B') {
// drive backward
btx = (BT.read());
motorsBackward(0, 250);
if (btx == 'O') {
motorsOff();
break;
}
}
while (bt_signal == 'L') {
// turn left
btx = (BT.read());
motorsTurnLeft(0, 200);
if (btx == 'O') {
motorsOff();
break;
}
}
while (bt_signal == 'R') {
// turn right
btx = (BT.read());
motorsTurnRight(0, 200);
if (btx == 'O') {
motorsOff();
break;
}
}
while (bt_signal == 'A') {
// line-tracking
btx = (BT.read());
line_tracking();
if (btx == 'O') {
motorsOff();
break;
}
}
while (bt_signal == 'S') {
// ultrasonic sensors
btx = (BT.read());
ultrasonic_sensors();
if (btx == 'O') {
motorsOff();
            break;
}
```

```
      }
    }
}
```

Funkcja przeznaczona do nasłuchiwanie poleceń wydawanych przez aplikację kliencką poprzez bluetooth. Dla każdego kodu wywołuje odpowiednią funkcję odpowiedzialną za daną funkcjonalność.
Otrzymanie kodu “O” za każdym razem wywołuje przerwanie aktualnej czynności i wyłączenie silników

3. **loop**

```
void loop() {
        listen_bluetooth();
 }
```

Funkcja pętli, wywoływana cały czas podczas działania Arduino. Znajdziemy w niej jedynie funkcję **listen_bluetooth** - robot po włączeniu cały czas czeka na rozkaz od aplikacji klienckiej.

4. **motorsOff**

```
void motorsOff() {
        // turn all the motors off
        digitalWrite(in1, LOW);
        digitalWrite(in2, LOW);
        digitalWrite(in3, LOW);
        digitalWrite(in4, LOW);
}
```

Wysłanie do wszystkich silników sygnały LOW, co powoduje ich wyłączenie.

5. **motorsForward**

```
void motorsForward(int period, int power) {
  // run all the motors in a forward direction

  // turn on motors
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);

  // set motors speed
  analogWrite(enA, power);
```

```
  analogWrite(enB, power);

   // if period is 0 motors will run for an unlimited period of time
   if (period != 0) {
         delay(period);
         motorsOff();
   }
 }
```

Funkcja odpowiedzialna za jazdę robota do przodu. Przyjmuje okres w jakim ma jechać pojazd w mikrosekundach oraz moc (od 0 do 255). Jeśli okres będzie równy 0 jazda będzie wykonywana bez końca, aż do sygnału przerwania.

6. motorsBackward, motorsTurnLeft, motorsTurnRight

```
void motorsBackward(int period, int power) {
  // run all the motors in a backward direction

  // turn on motors
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);

  // set motors speed
  analogWrite(enA, power);
  analogWrite(enB, power);

  // if period is 0 motors will run for an unlimited period of time
  if (period != 0) {
         delay(period);
         motorsOff();
  }
}

void motorsTurnRight(int period, int power) {
  // run the left motors only

  // turn on motors
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);

  // set motors speed
  analogWrite(enA, power);
  analogWrite(enB, power);
```

```
  // if period is 0 motors will run for an unlimited period of time
  if (period != 0) {
        delay(period);
        motorsOff();
  }
}

void motorsTurnLeft(int period, int power) {
  // run the right motors only

  // turn on motors
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);

  // set motors speed
  analogWrite(enA, power);
  analogWrite(enB, power);

  // if period is 0 motors will run for an unlimited period of time
  if (period != 0) {
        delay(period);
        motorsOff();
  }
}
```

Analogicznie do **motorsForward**, funkcja odpowiedzialna za jazdę robota do tyłu, w lewo i w prawo.

7. **IRleft, IRright, IRcenter**

```
int IRleft() {
  return digitalRead(leftIR);
}

int IRright() {
  return digitalRead(rightIR);
}

int IRcenter() {
  return digitalRead(centerIR);
}
```

Funkcje zwracające wartości poszczególnych czujników na podczerwień. 0 zwracane jest, gdy czujnik wykrywa pole czarne, 1 gdy wykrywa pole białe.

8. **IRleft, IRright, IRcenter**

```
void line_tracking() {
        while(IRleft() == 1 && IRcenter() == 0 && IRright() == 1) {
                btx = (BT.read());
                if (btx == 'O') {
                        break;
                }
                motorsForward(0,90);
        }
        while(IRleft() == 1 && IRcenter() == 1 && IRright() == 1) {
                btx = (BT.read());
                if (btx == 'O') {
                        break;
                }
                motorsForward(0,90);
        }
        while(IRleft() == 1 && IRcenter() == 1 && IRright() == 0) {
                btx = (BT.read());
                if (btx == 'O') {
                        break;
                }
                motorsTurnRight(0,160);
        }
        while(IRleft() == 0 && IRcenter() == 1 && IRright() == 1) {
                btx = (BT.read());
                if (btx == 'O') {
                        break;
                }
```

```
            motorsTurnLeft(0,160);
        }
        while(IRleft() == 0 && IRcenter() == 0 && IRright() == 1) {
            btx = (BT.read());
            if (btx == 'O') {
                break;
            }
            motorsTurnLeft(0,200);
        }
        while(IRleft() == 1 && IRcenter() == 0 && IRright() == 0) {
            btx = (BT.read());
            if (btx == 'O') {
                break;
            }
            motorsTurnRight(0,200);
        }
        while(IRleft() == 0 && IRcenter() == 0 && IRright() == 0) {
            btx = (BT.read());
            if (btx == 'O') {
                break;
            }
            motorsOff();
        }
}
```

Funkcja obsługująca funkcjonalnośc line-tracking. Rozróżniamy nastepujące akcje robota, w kolejności jak w kodzie:

1. Jazda do przodu - jeśli czujnik środkowy zwraca 0 (czarne), a pozostałe 1 (białe); lub jeśli wszystkie czujniki zwracają 1 (białe).
2. Delikatny skręt w lewo lub w prawo - jeśli czujnik środkowy i skrajny (lewy lub prawy) zwracają 1 (białe), a tylko jeden czujnik zwraca 0 (czarne).
3. Ostry skręt w lewo lub w prawo - jeśli tylko jeden skrajny czujnik zwraca 1 (białe) a reszta 0 (czarne).
4. Stój - jeśli wszystkie czujniki zwracają 0 (czarne). Sytuacja taka występuje np.: gdy robot podniesiony jest znad trasy