4 IMPLEMENTASI SISTEM

Dalam bab implementasi sistem ini akan dijelaskan mengenai penerapan metode, teori, dan *library* yang sudah ditulis pada bab sebelumnya.

Pada tugas akhir ini sistem terbagi menjadi tiga bagian, yaitu:

- Desain skenario untuk pengambilan dan pelatihan data yang dibuat dengan platform *Openvibe.*
- Aplikasi *client* untuk menerima data dari *Openvibe* yang dibuat dalam bahasa pemrograman C++ menggunakan Visual Studio 2013.
- Aplikasi *server* untuk menggerakkan motor mobil *rc* yang dibuat dalam bahasa pemroraman C++ menggunakan Arduino IDE.

4.1 Koneksi Antar Aplikasi Yang Dibuat

Aplikasi pertama yang dibuat adalah desain dan dibuat di dalam platform Openvibe yang terdiri dari enam desain atau *scenario* berbeda. *Openvibe* juga berfungsi untuk menghubungkan headset emotiv epoc sehingga dapat difungsikan untuk pengambilan data.

Scenario akan mengasilkan perintah untuk menggerakkan mobil rc, yang berupa label kelas (maju, kiri, dan kanan). Label ini akan dikirimkan ke *client* dengan menggunakan *virtual reality peripheral network (VRPN)* dari *box* yang sudah disediakan oleh *openvibe*.

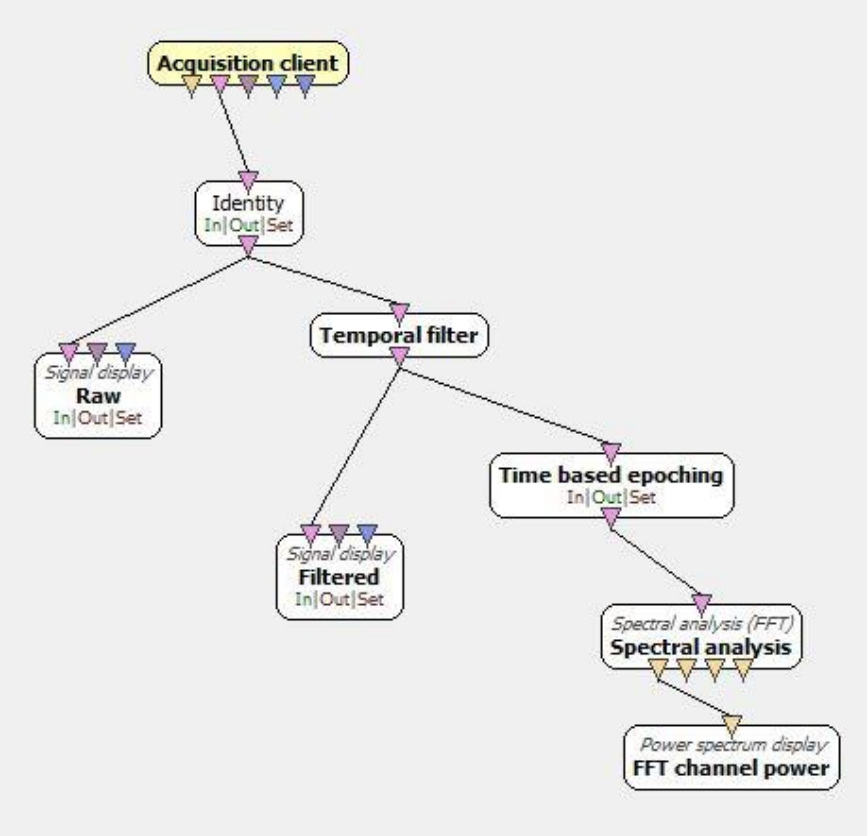
Kemudian *client* akan mengirimkan perintah ke *server* untuk menggerakkan motor mobil rc. Perintah ini dikirimkan melalui wifi yang sudah terhubung antara *client* dan mobil. Perintah ini diterima oleh modul *board* arduino yang sudah terprogram dengan aplikasi *server* yang dibuat.

4.2 Openvibe Design

Pada *openvibe*, *scenario* yang dibuat akan menggunakan *Steady State Visually Evoked Potentials (*SSVEP*)* untuk stimulasi user.

4.2 1 Desain Pertama

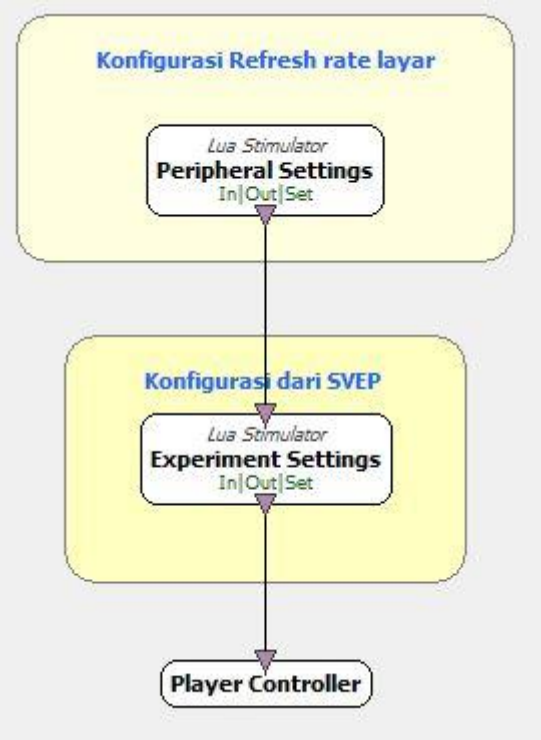
Desain Pertama yang dibuat adalah *acquisition-test.xml*. Fungsi dari desain ini adalah untuk memonitor apakah headset emotiv sudah siap untuk dipakai dan dapat menerima sinyal EEG dengan baik. Selain itu pada desain ini juga dapat digunakan untuk melihat bagaimana bentuk sinyal EEG setelah dilakukan *filtering* menggunakan *badpass-filter*, dan juga bentuk sinyal dalam *frekuensi domain* menggunakan *fast fourier transform.* Untuk Gambar desain dapat dilihat pada Gambar 4.1.



Gambar 4.1 Aquitation-test.xml

4.2.2 Desain Kedua

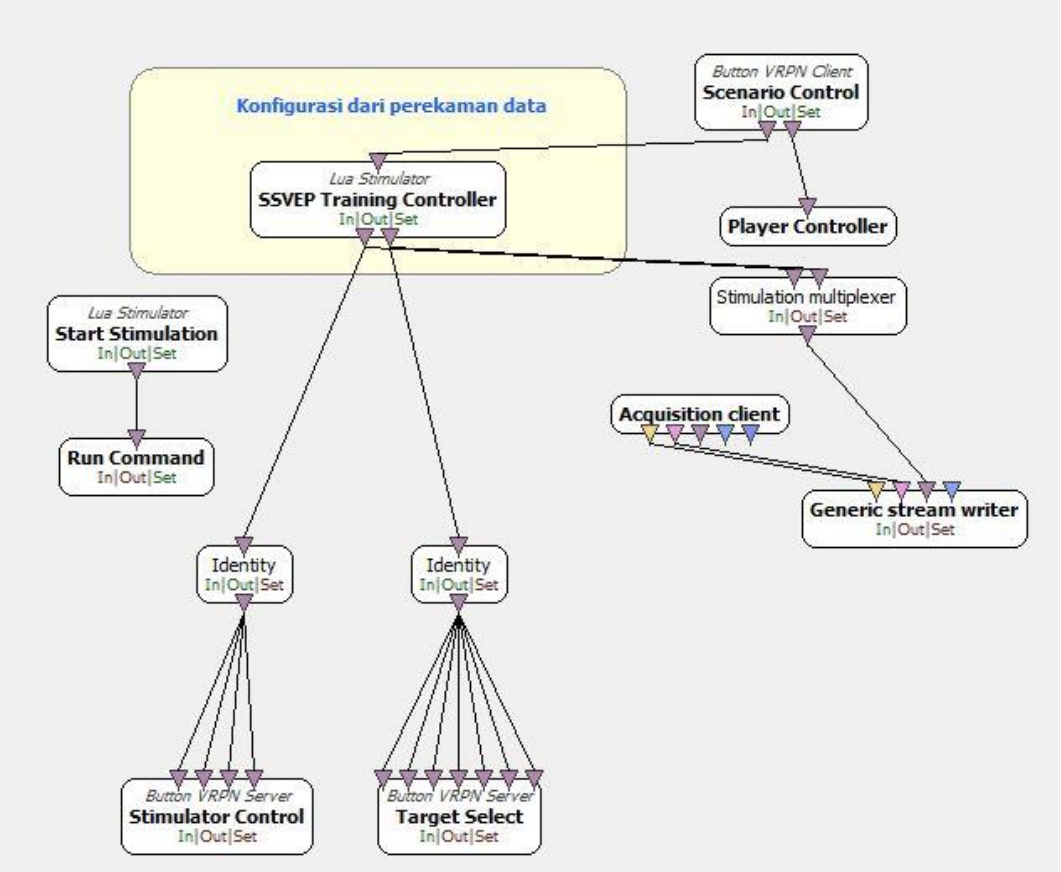
Desain kedua yang dibuat adalah *SSVEP-configuration.xml*. fungsi dari desain ini adalah untuk mengatur konfigurasi dari SSVEP. Desain skenario ini harus di jalankan terlebih dahulu sebelum bisa melanjutkan ke skenario sebelumnya. Hal ini dikarenakan pada skenario kedua ini akan menghasilkan *configuration* file yang nantinya akan di *load* pada skenarion selanjutnya. File ini berupa file .cfg untuk menyimpan konfigurasi dari aplikasi SSVEP. Untuk gambar desain dapat dilihat pada Gambar 4.2 pada halaman selanjutnya



Gambar 4.2 SSVEP-configuration.xml

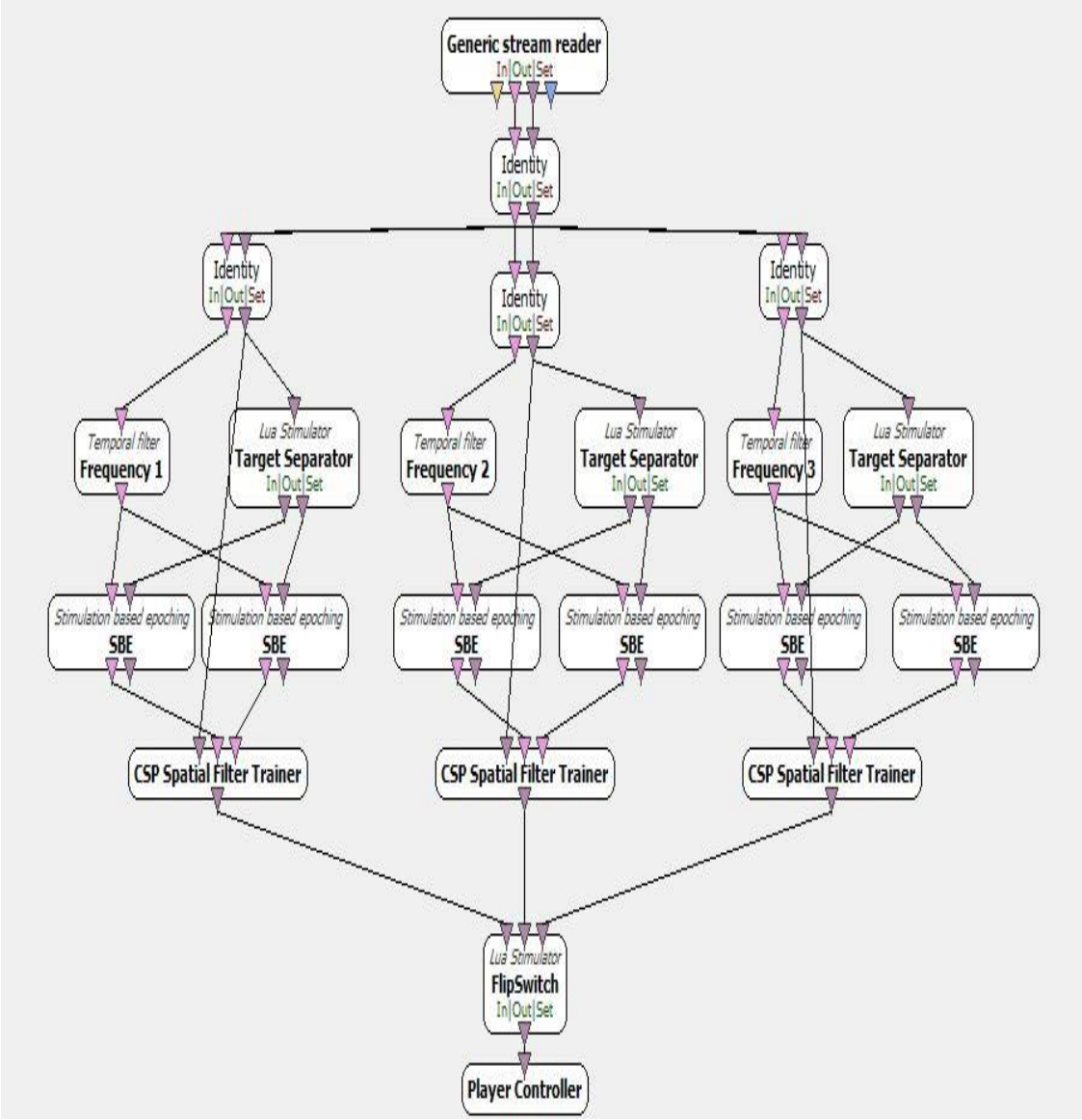
4.2.3 Desain Ketiga

Desain ketiga yang dibuat adalah *acquitation-training.xml*. Fungsi dari desain ini adalah untuk merekam data yang nantinya akan digunakan dalam peltihan. Dalam desain ini kita dapat mengatur banyak data yang akan diambil untuk tiap kelasnya dan urutan kelas dalam pengambilan data. Selain itu pengguna juga dapat mengatur *refresh rate* untuk setiap box. Hasil dari desain ini adalah file *raw data* yang ber ekstensi .ov. Setelah skenario dijalankan layar akan menampilkan box untuk SSVEP yang memantu pengguna dalam perekaman data. Untuk gambar desain dapat dilihat pada gambar 4.3 di halaman berikutnya.



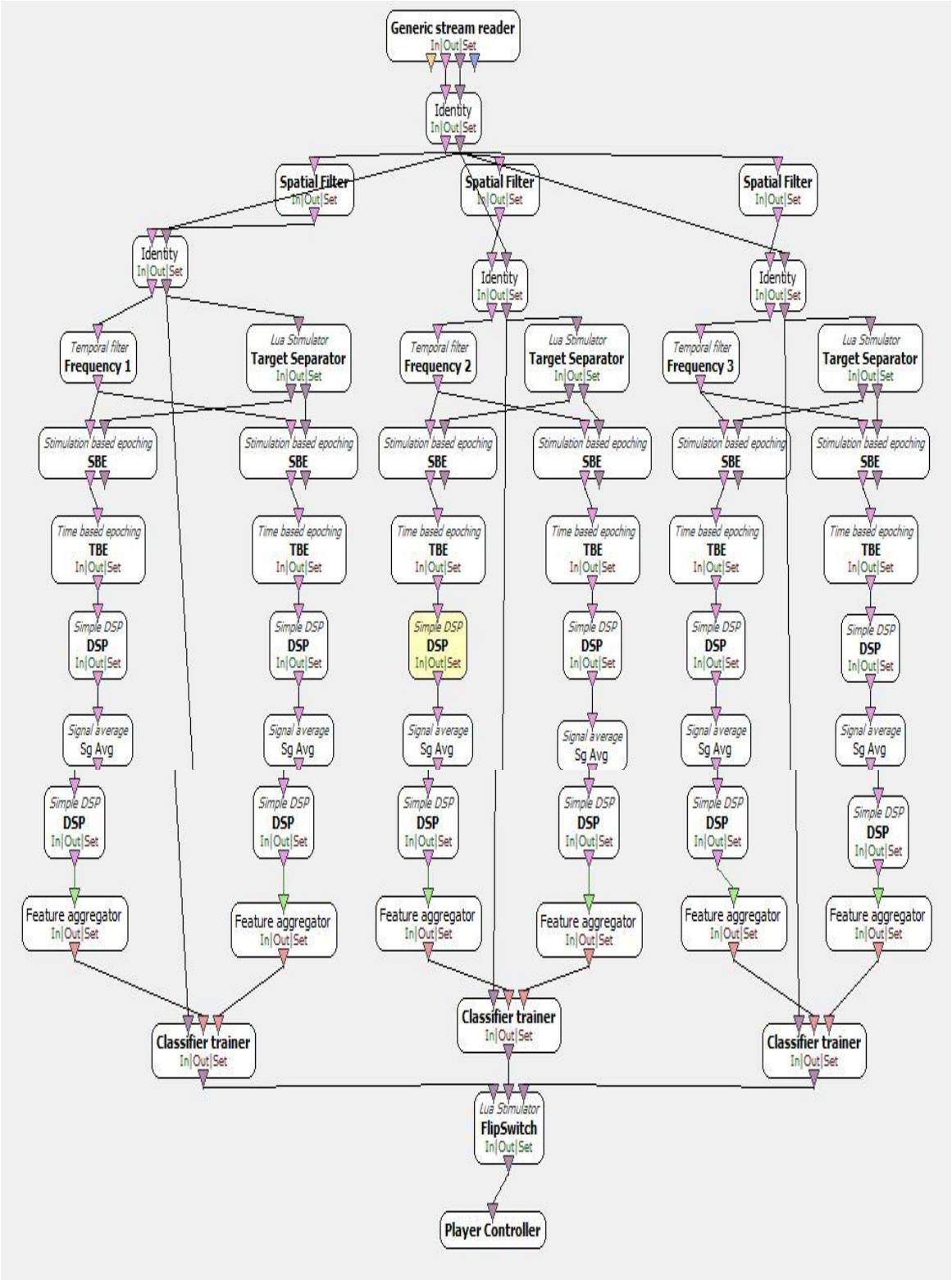
4.2.4 Desain Ke-empat

Desain ke-empat yang dibuat adalah *CSP-training.xml*. Fungsi dari desain ini adalah untuk melakukan *feature extraction* dari *raw data* dan juga memilih secara otomatis chanel data terbaik dari data yang diambil menggunakan desain sebelumnya. Dari desain ini akan menghasilkan konfigurasi untuk fitur dan memilih channel data secara otomatis pada tiap kelas yang ingin dikenali dengan inputan data dari desain sebelumnya (desain tiga). Untuk gambar desain dapat dilihat pada gambar 4.4 di halaman berikutnya.



4.2.5 Desain Ke-Lima

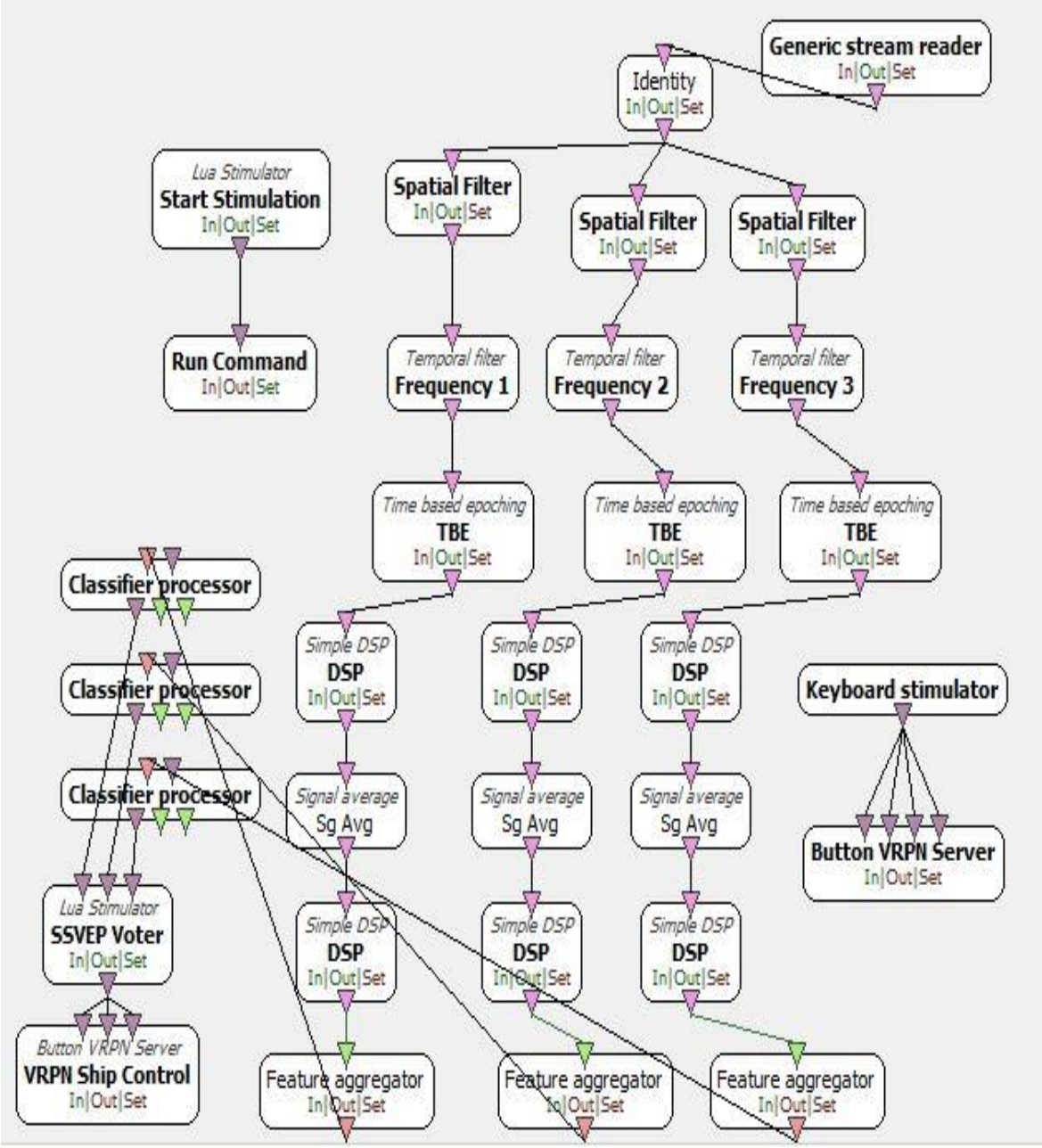
Desain ke-lima yang dibuat adalah classifier-training.xml. Fungsi dari desain ini adalah membuat *predict model* dengan cara melatih *raw datai* sehingga dapat diklasifikasikan sesuai kelasnya. Pada desain ini akan mengasilkan konfigurasi untuk tiap kelas (*predict model*) dengan inputan data dari desain sebelumnya (desain tiga dan empat).). Untuk gambar desain dapat dilihat pada gambar 4.5 di halaman berikutnya.



Gambar 4.5 Classification-training.xml

4.2.6 Desain ke-Enam

Desain ke-enam yang dibuat adalah *online-recognition.xml.* Fungsi dari desain ini adalah menghasilkan perintah untuk mobil rc. Perintah dihasilkan dengan memcocokkan sinyal EEG secara *online* dengan *predict model* yang dihasilkan dari desain sebelumnya. Selain pengenalan secara ofline pengguna juga bisa mneggunakan data *offline* pengenalan sinyal EEG. Untuk gambar desain dapat dilihat pada gambar 4.6



Berikut adalah penjelasan dari Box yang digunakan pada desain skenario :

- Acquisition Client : box ini berguna untuk menerima dan *mengoutputkan* sinyal yang diterima oleh openvibe dari headset emotiv.
- Identity : box ini berguna untuk menggandakan sinyal.
- Signal display : box ini berguna untuk menampilkan sinyak ke layar.
- Temporal Filter : box ini digunakan untuk melakukan *filtering (highpass,lowpass, butterworth filter)*
- Time based epoching : box ini digunakan untuk *windowing.*
- Spektral analysis : box ini digunakan untuk melakukan *fast fourier transform (FFT)*
- Power spektrum display : box ini digunakan untuk menampilkan hasil FFT ke layar
- Lua Stimulator : box ini digunakan untuk menjalankan *script .lua* yang telah dibuat.
- Player controller : box ini digunakan untuk mengontrol aplikasi (start, stop, dll)
- Stream writer : box ini digunakan untuk membuat file (merekam EEG menjadi file .ov)
- Stream reader : box ini digunakan untuk membaca file hasil dari box steam writer
- Button VRPN Server : box ini digunakan untuk mengirim label ke client
- CSP spatial filter trainer: box ini digunakan untuk melakukan feature ekstraction dari raw data dan menghasilkan file konfigurasi fitur.
- Simple DSP : box ini berguna untuk melakukan pemrosesan sinyal digital
- Feature agregator : box ini berguna untuk mengubah sinyal dari intputan menjadi vektor sehingga dapat digunakan untuk pelatihan
- Clasifier training : box ini berguna untuk melakukan training, dalam tugas akhir ini saya menggunakan SVM untuk metode training nya. Box ini menghasilkan file *predict model*
- Clasifier processor : box ini berguna untuk pengenalan kelas dengan menggunakan file *predict model* yang dihasilkan dari box clasifier training.

4.3 Implementasi Perangkat Lunak Untuk Client dan Server

Pembuatan aplikasi tugas akhir ini menggunakan bahasa pemrograman C++ dengan Microsoft Visual Studio 2013 sebagai *Integrated Development Environment* (IDE) untuk aplikasi client dan Arduino 1.0.6 sebagai IDE untuk aplikasi server. Alasan penggunaan bahasa C++ karena openvibe dapat di hubungkan dengan C++ dengan mudah daripada bahasa pemrogramman lainnya.

4.4 Library Yang Digunakan

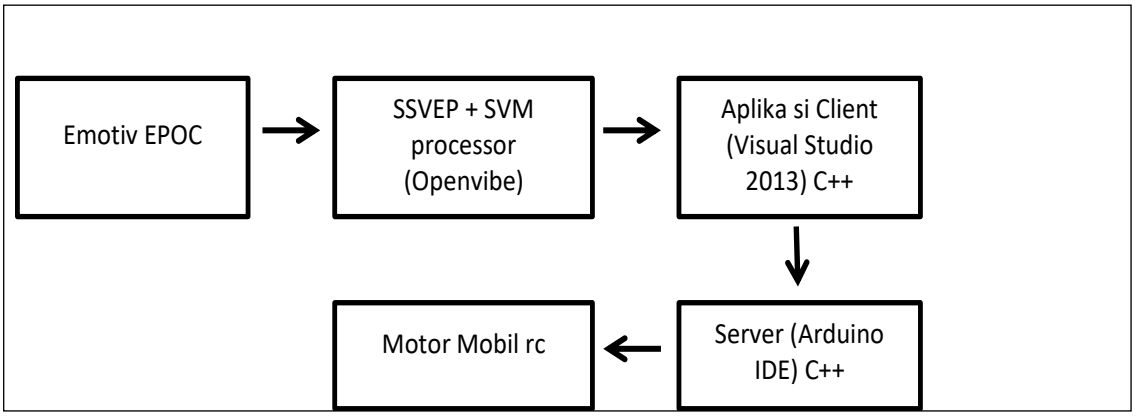
Pada pembuatan aplikasi tugas akhir ini, digunakan *library* vrpn.lib yang membantu dalam menerima object dari platform openvibe desainer. Dengan menggunakan *library* ini client dapat menerima object dengan memasukkan *peripheral name* yang sama di openvibe desainer serta client.

Selain itu, saya juga menggunakan ws2_32.lib yang merupakan *library* dari winsock2. Winsock2 adalah *library* yang dapat digunakan untuk *socket programming* di microsoft visual studio. Dengan *library* ini saya dapat membuat aplikasi client dan server sehingga dapat saling mengirim dan menerima data.

Dalam tugas akhir ini data yang dikirimkan oleh client berupa *struct* object yang berisikan gerak untuk motor mobil rc.

4.5 BCI System

Untuk diagram sistem yang akan digunakan pada tugas akhir ini dapat dilihat pada Gambar 4.7.



Gambar 4.7 Block diagram Sistem BCI

Pertama kali user akan diminta untuk berfikir untuk setiap kelas (arah kiri, kanan, maju, mundur), setiap kelas akan dilakukan perekaman selama 3 detik dengan

jumlah perulangan sebanyak 10 kali pada tiap kelasnya. Selanjutnya proses pengambilan data dimulai, disini saya menggunakan openvibe-acquisition-server yang telah disambungkan dengan Emotiv premium sdk yang telah terinstall. Data yang direkam memiliki *sampling rate* 128 tiap detiknya dan disimpan dengan bentuk file *.ov* yang merupakam file *native* dari Openvibe.

Pada tahap *preprocessing* dilakukan penghilangan noise / *denoising* untuk gerakan otot yang tidak perlu (seperti berkedip, gerakan mulut, dll) dengan cara *high-pass* dan *low-pass filter*. Setelah raw data bebas dari *noise*, dilakukan *windowing* untuk mengambil data yang penting dengan tepat yang nantinya digunakan untuk peng ekstrakkan fitur menggunakan CSP. Hasil dari CSP adalah vektor pada setiap chanel dari headset emotiv epoc (emotiv epoc memiliki 14 chanel sensor). Vektor ini digunakan sebagai data pelatihan pada SVM, hasil pelatihan dari SVM menghasilkan *predict model* untuk pengenalan sinyal secara online di aplikasi dan menghasilkan output pergerakan dengan tepat.

Setelah aplikasi mengeluarkan *output* untuk suatu kelas, maka mobil diharapkan bergerak sesuai arah yang disebutkan. *Output* sebelumnya akan di proses dengan aplikasi sehingga dapat memberikan perintah untk menggerakkan motor mobil rc.

4.6 Segmen Program Dari Perangkat Lunak

Pada tugas akhir ini, segmen dari program akan dibagi menjadi dua, yaitu segmen untuk client dan untuk server.

4.6.1 Segmen Client

Pada segment client ini aplikasi dibuat menggunakan bahasa pemrograman C++ dan Visual Studio 2013 sebagai IDE. Untuk tabel fungsi dan prosedur dapat dilihat pada tabel 4.1 dibawah ini

Tabel 4.1 Daftar fungsi untuk aplikasi client

Segmen Program	Nama Fungsi	Keterangan Fungsi
4.1	VRPNbuttonserver	Mempersiapkan variabel untuk menerima data dari openvibe vrpn button server
4.2	BindingVRPNbutton	Menghubungkan VRPNbutton yang sudah dibuat dengan fungsi

		VRPncallback yang dibuat
4.3	VrpnCallback	Proses yang dilakukan setiap kali ada data yang diterima oleh client
4.4	UDPconstruct	Membuat koneksi UDP
4.5	SendMessage	Mengirimkan paket data ke server melalui UDP

- Persiapkan VRPN Button Server : Pada tahapan ini akan digunakan untuk membuat koneksi antara openvibe dan client melalui peripheral yang sudah dibuat sebelummnya. Untuk *source code* dari segmen client dapat dilihat pada Segmen 4.1.

```
/* VRPN Button object */
      vrpn_Button_Remote* VRPNButton;
      VRPNButton = new vrpn_Button_Remote(buttonDevice);
      vrpn_Button_Remote* VRPNButton2;
      VRPNButton2 = new vrpn_Button_Remote(buttonDevice2);
```

Segmen 4.1 source code vrpn obejct

- Binding VRPN Button : Pada tahapan ini dilakukan untuk menghubungkan object vrpn_button yang dibuat dengan fungsi *callback* yang dibuat. Untuk *source code* dari segmen client dapat dilihat pada Segmen 4.2

```
      /* Binding of the VRPN Button to a callback */
      VRPNButton->register_change_handler(&running, vrpn_button_callback);
      VRPNButton2->register_change_handler(&running, vrpn_button_callback2);
      //to free memory after use
      VRPNButton->unregister_change_handler(&running, vrpn_button_callback);
      VRPNButton2->unregister_change_handler(&running,
      vrpn_button_callback2);
      delete VRPNButton;
      delete VRPNButton2;
```

Segmen 4.2 Source code untuk Binding vrpn button

- VRPN callback : vrpn_callback adalah sebuah fungsi yang akan

```
void VRPN_CALLBACK vrpn_button_callback2(void* user_data, vrpn_BUTTONCB but-
ton){
      // std::cout << "Button ID : " << button.button << " / Button State : "
<< button.state << std::endl;
      std::cout << "keyboard ke :" << count_keyboard << "|";
      duration = (std::clock() - start) / (double)CLOCKS_PER_SEC;
      std::cout <<" waktu : "<< duration <<"|";
      count_keyboard++;
      //isi variabel car
      rc_data.speed = 0;
      rc_data.turn = 0;
      rc_data.flags = 0;
      rc_data.cmd = 0;
      if (button.button == 0){
            cout << " kelas :maju\n";
            //strcpy(message, "maju");
            rc_data.speed = -700;
            rc_data.turn = -700;
            rc_data.cmd = 1;
            //send_messagae();
      }else if (button.button == 1){
            cout << " kelas :kiri\n";
            rc_data.turn = -1023;
            rc_data.speed = 1023;
            rc_data.cmd = 1;
            //send_messagae();
      }
else if (button.button == 2){
            cout << " kelas :kanan\n";
            rc_data.turn = 1023;
            rc_data.speed = -1023;
            rc_data.cmd = 1;
            //send_messagae();
      }else if (button.button == 3){
            cout << " kelas :mundur\n";
            rc_data.turn = 1023;
            rc_data.speed = 1023;
            rc_data.cmd = 1;
            //send_messagae();
      }
}
```

dipanggil setiap ada dat yang diterima dari box vrpn yang ada di openvibe. Data berupa button.id dan button.state. Untuk *source code* dari segmen client dapat dilihat pada Segmen 4.3

- UDP untuk pengiriman data : Pengiriman data ke server menggunakan *library winsock2* , sehingga data dikirimkan melalui *socket* yang terhubung dengan *interner protocol (IP)* dan juga

socket dari board Wemos D1. Untuk *source code* dari segmen client dapat dilihat pada Segmen 4.4

Segmen 4.4 Source code untuk UDP client

```
//include nya
#pragma comment(lib,"ws2_32.lib") //Winsock Library
#define SERVER "192.168.1.115"  //ip address of udp server
#define BUFLEN 512  //Max length of buffer
#define PORT 4320   //The port on which to listen for incoming data
//udp variable
struct sockaddr_in si_other;
int s, slen = sizeof(si_other);
char buf[BUFLEN];
char message[BUFLEN];
WSADATA wsa;
//di main functionnya
//create socket
      if ((s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == SOCKET_ERROR){
            printf("socket() failed with error code : %d",
WSAGetLastError());
            exit(EXIT_FAILURE);
      }
      //setup address structure
      memset((char *)&si_other, 0, sizeof(si_other));
      si_other.sin_family = AF_INET;
      si_other.sin_port = htons(PORT);
      si_other.sin_addr.S_un.S_addr = inet_addr(SERVER);
      closesocket(s);
      WSACleanup();
```

- Data yang dikiriman berupa *Struct Data* yang berisikan variabel yang digunakan untuk menggerakkan motor. Untuk *source code* dari segmen client dapat dilihat pada Segmen 4.5

```
void send_messagae(){
      //cout << "mulai kirim\n";
      if (sendto(s, (char *)&rc_data, sizeof(rc_data), 0, (struct sockaddr *)
&si_other, slen) == SOCKET_ERROR){
            printf("sendto() failed with error code : %d",
WSAGetLastError());
            exit(EXIT_FAILURE);
      }
      //receive a reply and print it
      //clear the buffer by filling null, it might have previously received
data
      memset(buf, '\0', BUFLEN);
      //try to receive some data, this is a blocking call
      if (recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *) &si_other, &slen)
== SOCKET_ERROR){
            printf("recvfrom() failed with error code : %d",
WSAGetLastError());
            exit(EXIT_FAILURE);
      }
}
```

Segmen 4.5 Source code untuk *function* send_message()

4.3.2 Segmen Server

Pada segment server ini aplikasi dibuat menggunakan bahasa pemrograman C++ dan Arduino sebagai IDE. Untuk tabel fungsi dan prosedur dapat dilihat pada tabel 4.2 dibawah ini

Tabel 4.2 Fungsi untuk aplikasi server

Segmen Program	Nama Fungsi	Keterangan Fungsi
4.6	UDPsetup	Membuat koeneksi UDP
4.7	UDPloop	Proses yang dilakukan setiap kali ada data paket yang diterima dari client
4.8	SetupServer	Mengatur Router dan soket yang digunakan untuk koneksi dengan client
4.9	Motor	Mengatur Pin yang digunakan untuk motor roda mobil rc dan proses gerak motor

- UDP Setup : Pada tahapan digunakan untuk menginisialisasi koneksi UDP dengan *client.* Pada tahapan ini kita harus mengatur soket yang digunakan. Untuk *source code* dari segmen client dapat dilihat pada Segmen 4.6

Segmen 4.6 Source code untuk UDPSetup()

```
//variabel untuk UDP Server
WiFiUDP Udp;
unsigned int localUdpPort = 4320;  // local port yang digunakan
byte incomingPacket[255];  // buffer untuk file yang diterima
long time_udp_call;
bool udp_call;
//---------------------------------------
void UDPSetup(){
  Udp.begin(localUdpPort);
  udp_call = false;
}
```

- UDP Loop : Pada tahapan ini digunakan untuk memproses setiap kali ada paket data yang diterima dari *client* dan juga mengirimkan balik paket data ke *client*. Untuk *source code* dari segmen client dapat dilihat pada Segemen 4.7 di halaman selanjutnya

```
void UDPLoop(){
  int packetSize = Udp.parsePacket();
//menyiapkan untuk menyimpan data dari client
  if( packetSize ){
    TRCcar rc_car;
    // menerima  UDP packets dari client
    int len = Udp.read(incomingPacket, 255);
    Serial.printf("UDP: %s\n", incomingPacket);*/
    for(int i=0;i<len;i++){
      Serial.println( "UDP[" + String(i) + "]: " + String( incomingPacket[i]
) );
      byte b = incomingPacket[i];
      uint8_t *p = (uint8_t*)(&rc_car) + i;
      memcpy(p, &b, 1);
    }
    Serial.println( "sizeof= " + String( sizeof(TRCcar) ) );
    Serial.println( "rc_car.speed= " + String( rc_car.speed ) );
    Serial.println( "rc_car.turn= " + String( rc_car.turn ) );
    // Mengirim balik ke clien dan memberitahu jika data sudah diterima
    Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
    Udp.write( "ESP8266" );
    Udp.endPacket();
    time_udp_call = millis();
    udp_call = true;
    MotorSpeed(abs(rc_car.speed), rc_car.speed > 0 ? LOW : HIGH );
    MotroTurn(rc_car.turn);
  }
  if( udp_call && time_loop - time_udp_call > 200 ){
    MotorSpeed(0);
    MotroTurn(0);
    udp_call = false;
  }
}
```

Segemen 4.7 Source code untuk UDPLoop()

- Setup server : pada tahapan ini digunakan untuk menginisialiasai program dari server, seperti memasukkan nama SSID wifi dan juga password wifi yang digunakan. Dan menampilkan di serial status dari koneksi board

  Dengan wifi. Untuk *source code* dari segmen client dapat dilihat pada Segemen 4.8

```
#define WIFI_SSID "Elektro-Petra"  // isi dengan data ssid wifi dan password
#define WIFI_PASS "petra12345678"
Void Setup(){
      WiFi.mode(WIFI_STA);
       WiFi.begin(WIFI_SSID, WIFI_PASS);
        Serial.print("Connecting to WIFI");
        // tunggu koneksi dengan wifi
        while( WiFi.status() != WL_CONNECTED ) {
          delay(500);
          Serial.print(".");
        }
        Serial.println("");
        Serial.print("Connected to ");
        Serial.println( WIFI_SSID );
        Serial.print("IP address: ");
        Serial.println(WiFi.localIP());
}
```

Segemen 4.8 Source code untuk Setup()

- Motor : Pada tahapan digunakan untuk mengontrol pergerakan motor mobil rc. . Untuk *source code* dari segmen client dapat dilihat pada Segemen 4.9

Segemen 4.9 Source code untuk motor

```
//mengatur pin untuk motor
#define PIN_MOTOR_A_IA D2   //motor kiri
#define PIN_MOTOR_A_IB D3
#define PIN_MOTOR_B_IA D5   //motor kanan
#define PIN_MOTOR_B_IB D6
//variabel
int motor_speed;
bool motor_dir;
int force_turn;
void MotroSetupA(){
  pinMode(PIN_MOTOR_A_IA, OUTPUT);
  pinMode(PIN_MOTOR_A_IB, OUTPUT);
  digitalWrite(PIN_MOTOR_A_IA, LOW);
  digitalWrite(PIN_MOTOR_A_IB, LOW);
  motor_speed = 0;
  motor_dir = LOW;
  delay(100);
}
void MotroSetupB(){
  force_turn = 800;
  pinMode(PIN_MOTOR_B_IA, OUTPUT);
  pinMode(PIN_MOTOR_B_IB, OUTPUT);
  digitalWrite(PIN_MOTOR_B_IA, LOW);
  digitalWrite(PIN_MOTOR_B_IB, LOW);
  delay(100);
}
```

4.7 Segmen Hardware Motor

Segmen ini menjelaskan bagaimana parameter yang dikirim oleh client dan server untuk menggerakkan motor mobil remote control

4.7.1 Segmen Server

Pada tabel 4.3 dibawah dapat dilihat untuk parameter penggerak motor degan server.

Tabel 4.3 Segmen Server Motor

Arah	Motor A		Motor B	
	1A	1B	1A	1B
Kiri	Analog write (Power)	Digital write (Low)	-	-
Kanan	-	-	Analog write (Power)	Digital write (High)
Maju	Analog write (Power)	Digital write (Low)	Analog write (Power)	Digital write (Low)
Mundur	Analog write (Power)	Digital write (High)	Analog write (Power)	Digital write (High)

4.7.2 Segmen Client

Pada tabel 4.4 dibawah dapat dilihat untuk parameter menggerakkan motor degan client.

Tabel 4.4 Segmen Client Motor

Arah	Motor A (Power)	Motor B(power)
Kiri	+	-
Kanan	-	+
Maju	-	-
Mundur	+	+