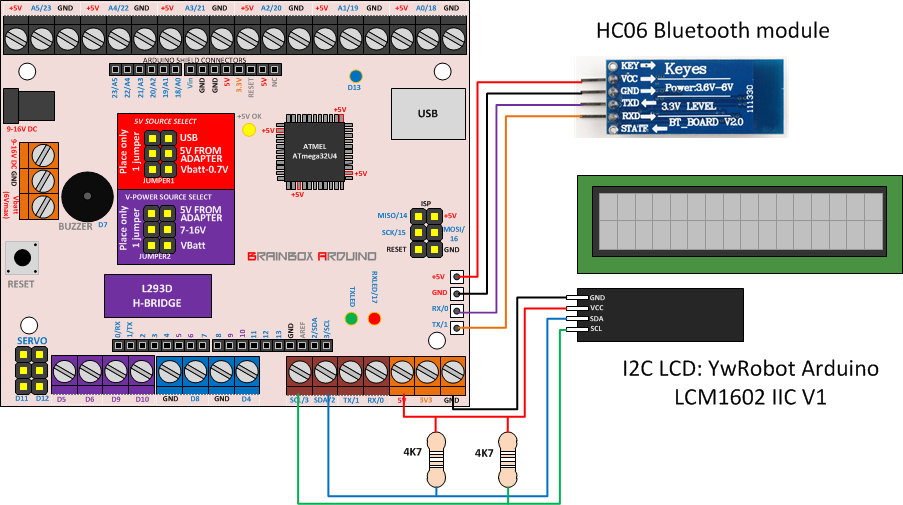
# **Nascholing sturen en meten met ARduino en Appinventor – versie 2021**

## BASISPROGRAMMA

**L:xxx;R:xxx;**



**xxx = data 0-1023**

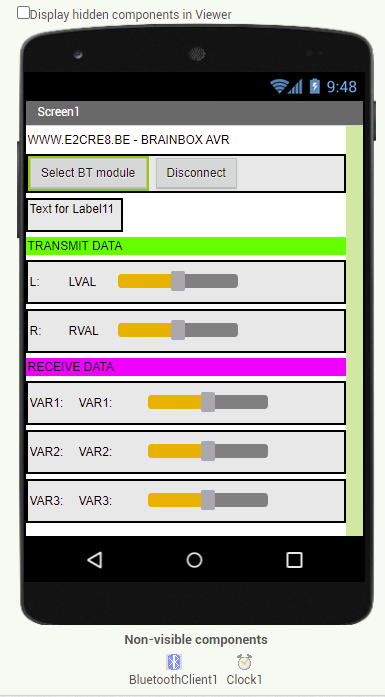
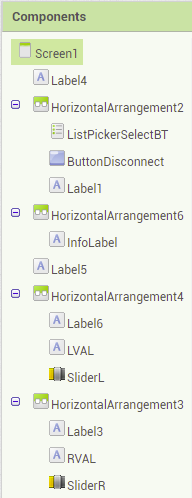
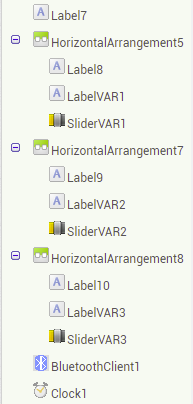
**yyy = data 0-1023**

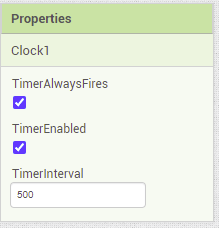
**zzz = data 0-1023**

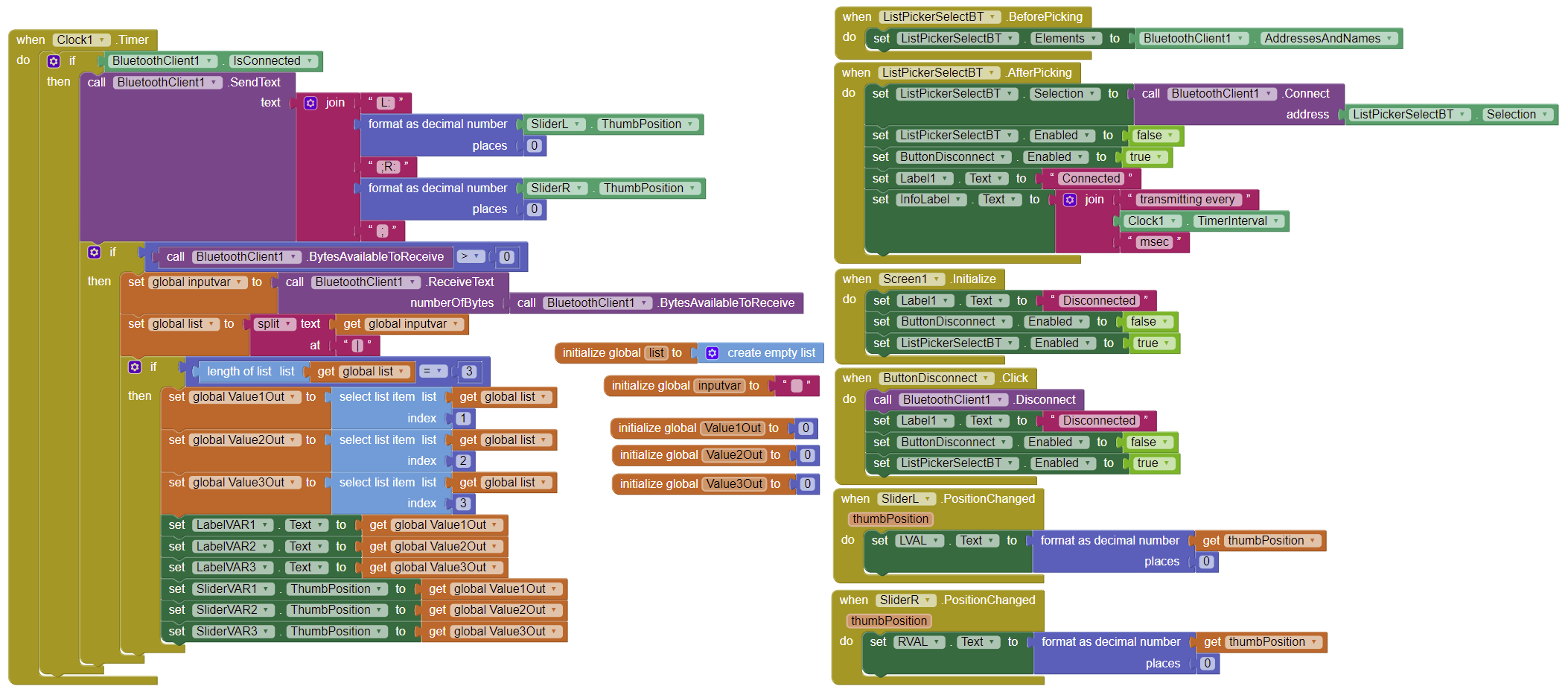
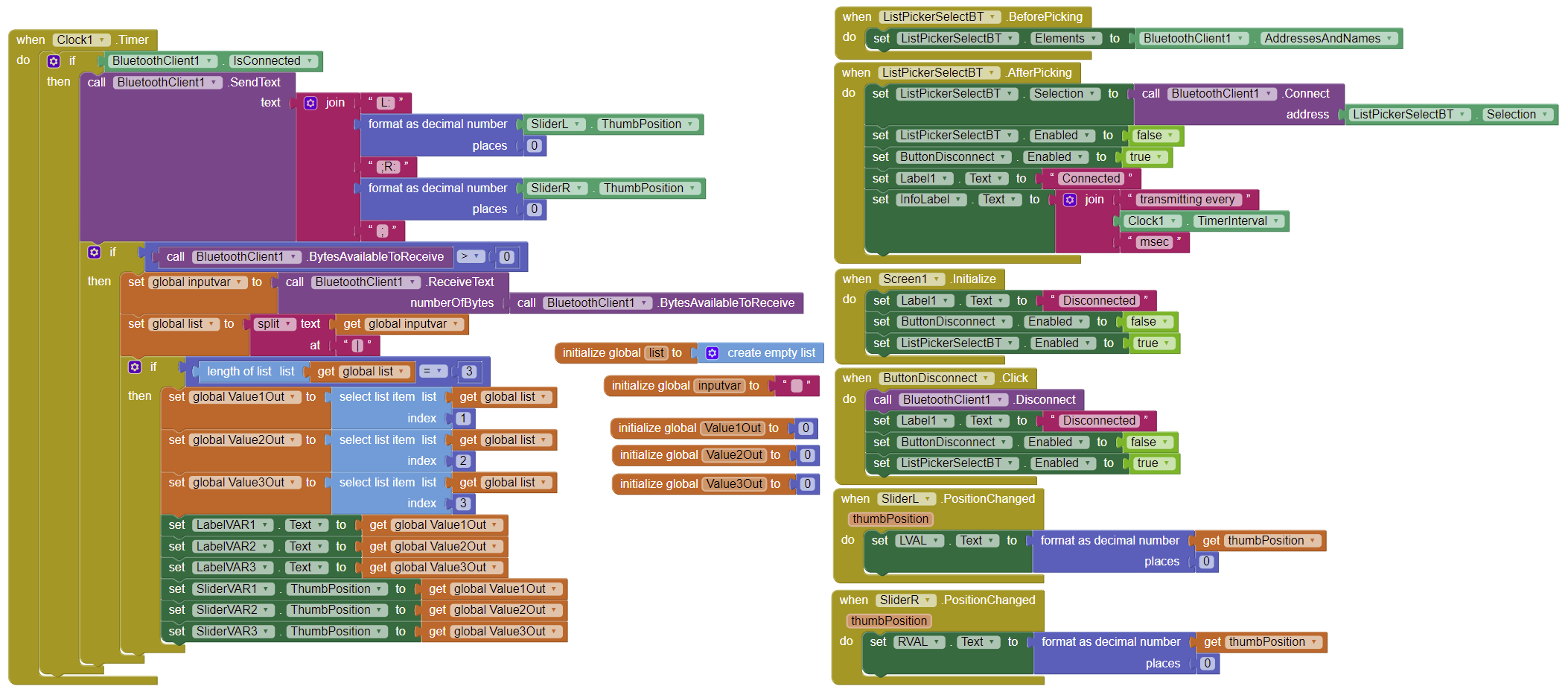
**xxx = data 0-255**

xxx|yyy|zzzz|

@ timerinterval x 2







// code written by Bart Huyskens  25/10/2021  for www.e2cre8.be

// hardware: Brainbox AVR (www.e2cre8.be >> Brainbox AVR)(This is a rugged Arduino Leonardo) ; HC06 Bluetooth module www.e2cre8.be ; Android Smartphone ; I2C LCD 4x20 www.e2cre8.be

// software: Arduino IDE 1.6.11 ; Appinventor

// Principle of communication:

// The app sends data to the uC in this format: "L:xxx;R:xxx;" where xxx is data between 0-255.

// The speed is determined by the TimerInterval of Timer1 in Appinventor and this needs to be identical to the variable "TimerIntervalAppinventor" in this program

// Speeds up to 100msec seem to be OK, but if you do not need this speed, use 1000msec

// The uC waits checks for incoming data at half of "TimerIntervalAppinventor" -time to be sure to not miss to much

// The uC send 3 random values to the app, every 2x "TimerIntervalAppinventor". This needs to be double the time for appinventor to be ready to receive the next data

#define TimerIntervalAppinventor 500      // this value needs to be equal to the value of the timer in Appinventor

                                         // tested & works pretty stable at 100msec, but if you don't need the speed, set it at 500msec both here and in Appinventor

#define tx 1          // pin number for TX pin

#define rx 0          // pin number for RX pin

int NrOfCycles;       // to slow down the transmission of data

int RValue = 0;

int LValue = 0;

int RValueOld = 0;

int LValueOld = 0;

int i = 0;

int z = 0;

bool connectionOK = 0;

char inSerial[13] = "";  // Array of characters (is not a string)- used to read the received data

int LValueArray[4];

int RValueArray[4];

String myString;        // used to do string manipulations on the received data

void setup()

{

**Serial**.begin(9600);             //For Serial monitor

**Serial1**.begin(9600);            // comms with HC06 - baudrate 9600

 pinMode(tx, OUTPUT);

 pinMode(rx, INPUT);

 while (**Serial1**.available())         //clear serial RX buffer

 {

   char t = **Serial1**.read();

 }

}

void loop()

{

 // Set up 3 variables to be sent

 int VAL1 = analogRead(A0);          // TX data 3 potmeters at A0, A2 & A4

 int VAL2 = analogRead(A2);

 int VAL3 = analogRead(A4);

 delay(TimerIntervalAppinventor / 2);     // timerInterval of Appinventor devided by 2, to be ready to receive all incoming data

 NrOfCycles = NrOfCycles + 1;             // transmission of data can only execute after 4 receive cycles

 //\*\*\*RECEIVING DATA \*\*\*//

 if (**Serial1**.available() > 0)        // If receiving serial data from HC06

 {

   connectionOK = 1;            // only start sending data after reception of first data -else uC crashes

   i = 0;

   while (**Serial1**.available()) // while receiving serial data from HC06

   {

     inSerial[i] = **Serial1**.read();   // load incoming bytes in array

     i++;                            // increment i

   }

 }

**//\*\*\*TRANSMITTING DATA \*\*\*\*//**

 if ((NrOfCycles >= 4) && connectionOK)       // transmission of data can only execute after 4 receive cycles for Appinventor to be ready to receive data

 {

**Serial1**.print(VAL1);     // send varialble VAL1 via Serial1 to HC06

**Serial1**.print('|');      // send '|' character - appinventor uses this to split the data

**Serial1**.print(VAL2);

**Serial1**.print('|');

**Serial1**.print(VAL3);

1023|1023|1023|

**Serial1**.print('|');

   NrOfCycles = 0;

 }

**//\*\*\*\*DECODING & FILTERING OF INCOMING DATA\*\*\*//**

   myString = String(inSerial);          // convert array to string to be able to use string manipulation instructions (indexOf...)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **L** | **:** | **2** | **5** | **5** | **;** | **R** | **:** | **4** | **8** | **;** |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **LValueOld** | **LValue** | **Lpos** | **m** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

 int Lpos = myString.indexOf('L');  // find the position of character 'L'

 if (Lpos == -1)                  // if Lpos = -1 then Lpos is not found

 {

   while (**Serial1**.available())

   {

     char t = **Serial1**.read();

   }

 }

 else

 {

   int m = 0;

   LValueOld = LValue;

   LValue = 0; // reset L value

   while ((myString.charAt(Lpos + 2 + m))!=;')

            //convert characters into RValue numbers untill character ';'

   {    // start 2 positions after character 'R'

     LValue = LValue \* 10 + ((myString.charAt(Lpos + 2 + m)) - 48);

//char '0' has ascii value 48

     m++;

   }

       if ((LValue > 255) || (LValue < 0)) LValue = LValueOld;

// protection from corrupt data

       // Accept data only if last 3 value were identical

|  |  |
| --- | --- |
| LValueArray[0] |  |
| LValueArray[1] |  |
| LValueArray[2] |  |
| LValueArray[3] |  |

       LValueArray[3] = LValueArray[2];

       LValueArray[2] = LValueArray[1];

       LValueArray[1] = LValueArray[0];

       LValueArray[0] = LValue;

if ((LValueArray[2] == LValueArray[1]) && (LValueArray[1] == LValueArray[0])) LValue = LValueArray[0];

else LValue =  LValueArray[3];

int Rpos = myString.indexOf('R');     // find the position of character 'R' in the string

   m = 0;

   RValueOld = RValue;

   RValue = 0;                      // Reset R Value

   while ((myString.charAt(Rpos + 2 + m)) != ';')     //convert characters into RValue numbers untill character ';'

   { // start 2 positions after character 'R'

     RValue = RValue \* 10 + ((myString.charAt(Rpos + 2 + m)) - 48); //char '0' has ascii value 48

     m++;

   }

       if ((RValue > 255) || (RValue < 0)) RValue = RValueOld;   // protection from corrupt data

       // Accept data only if last 3 values were identical

       RValueArray[3] = RValueArray[2];

       RValueArray[2] = RValueArray[1];

       RValueArray[1] = RValueArray[0];

       RValueArray[0] = RValue;

         if ((RValueArray[2] == RValueArray[1]) && (RValueArray[1] == RValueArray[0])) RValue = RValueArray[0];

         else  RValue =  RValueArray[3];

**// print op Seriele monitor**

   if (connectionOK != 1)

   {

**Serial**.println("NOT CONNECTED TO HC06");

   }

   else

   {

**Serial**.print(myString);               // Laat ontvangen string zien

**Serial**.print("\t");                   // Serial monitor - TAB

**Serial**.print(LValue);                 // Serial monitor Value of L

**Serial**.print("\t");                   // Serial monitor - TAB

**Serial**.print(RValue);                 // Serial monitor Value of R

**Serial**.print("\t");                   // Serial monitor - TAB

**Serial**.print(VAL1);                   // Laat verzonden string zien

**Serial**.print('|');                    // send '|' character - appinventor uses this to split the data

**Serial**.print(VAL2);

**Serial**.print('|');

**Serial**.print(VAL3);

**Serial**.print('|');

**Serial**.print("\t");                   // Serial monitor - TAB

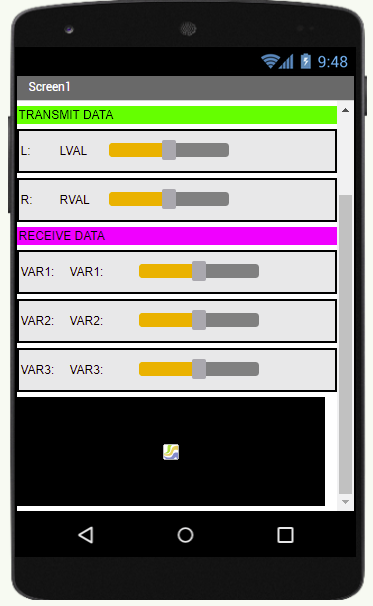
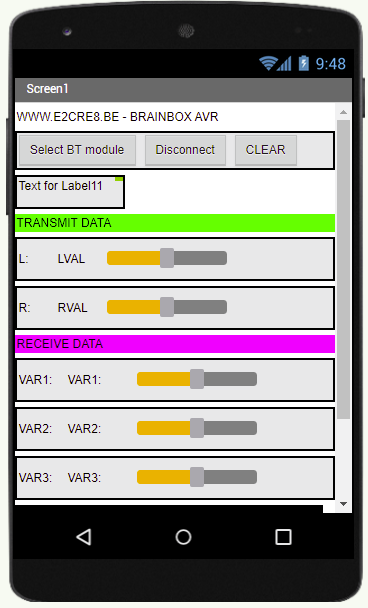
**Serial**.println("CONNECTED TO HC06");

   }

 }

}

## Werken met grafieken



Het doel van deze APP is om onze 3 meetwaarden grafisch voor te stellen in functie van de tijd. Een beetje zoals een oscilloscoop of datalogger. We moeten niet verwachten dat we identieke snelheden kunnen behalen als bij oscilloscopen, maar voor traag varierende signalen is het grote voordeel van deze APP dat we deze methode kunnen uitbreiden tot vele kanalen, waar een scoop bijna altijd beperkt blijft tot 2 kanalen.

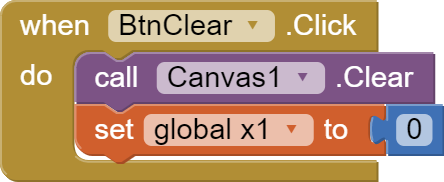
Het bovenste stuk van de schermopbouw blijft bijna identiek aan het het basisprogramma.Enkel de ‘Clear- Button’ is er hier bijgekomen. Met deze ‘Clear-button’ wissen we de grafieken op het scherm en zetten we de cursor terug links.

Onderaan het scherm hebben we een ‘Canvas’ gezet, met als ‘Backgroundcolor’ zwart. De ‘Width’ zetten we op ‘FillParent’ om dit canvas over de volledige breedte te verspreiden en de Height zetten we op 105 pixels om voldoende ruimte te hebben voor de meetwaarden die kunnen varieren tussen 0 en 1023. We zullen de meetwaarden dus moeten delen met een factor 10 om op het canvas te passen. De X-as van dit canvas is de horizontale as, de Y-as de verticale. Het nulpunt ligt linksboven.

Alle verdere instellingen gebeuren in het programma zelf.



We initialiseren om te beginnen 4 bijkomende variabelen die we allemaal op 0 zetten. X1 gebruiken we om de X-as te beïnvloeden en Y1, 2 en 3 gebruiken we om de 3 meetwaarden op de Y-as weer te geven.

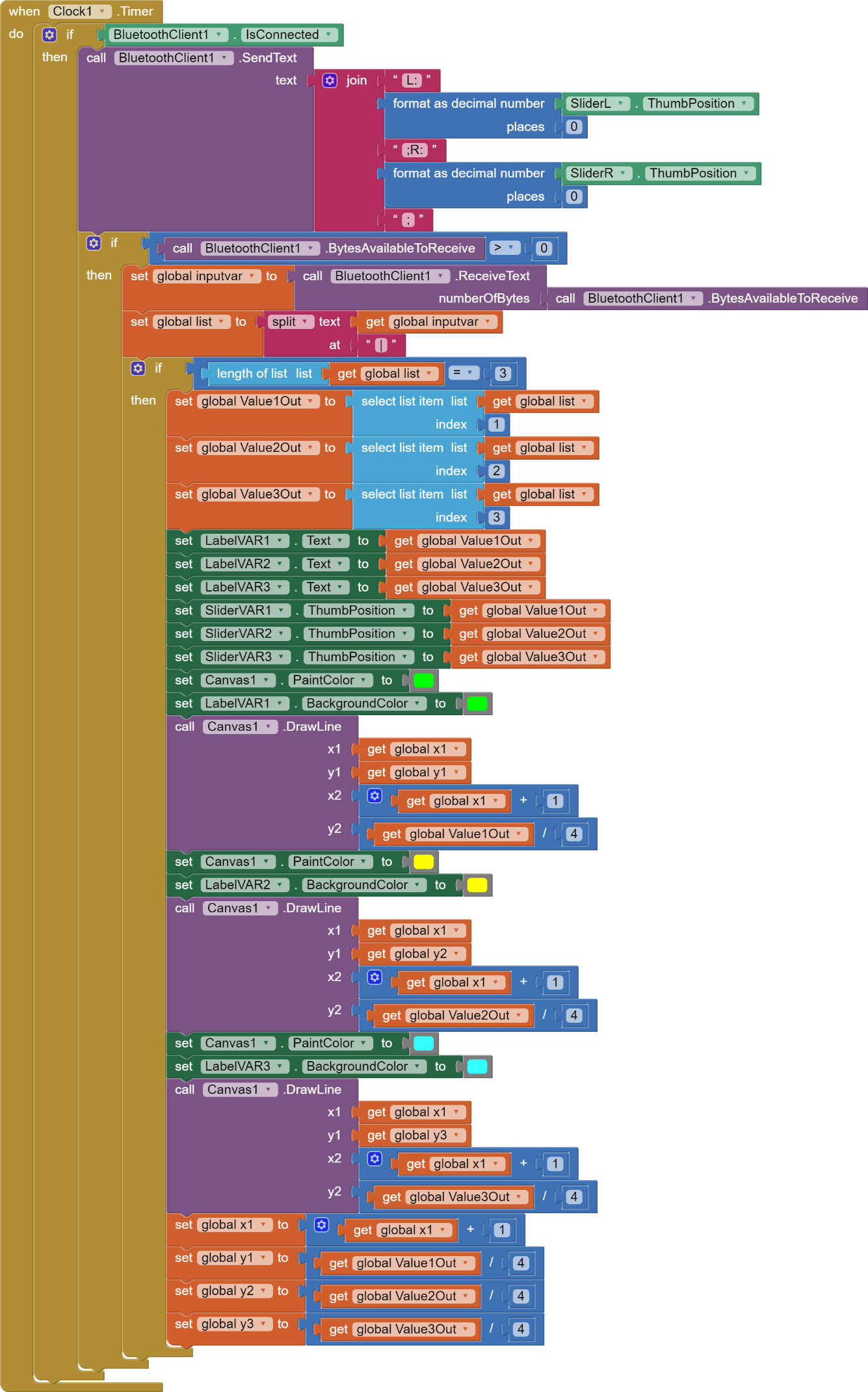


Wanneer er de Button met de CLEAR tekst – gedrukt wordt, dan wordt het canvas gewist en dan wordt de variabele X1 ook terug op 0 gezet

Het programma zelf wordt ook weer getriggerd door een Timer die deze block met instructies elke 500msec aanroept. Enkel het onderste deel van deze instructies is nieuw – al de rest is identiek aan ons basisprogramma.

We zetten de paintcolor van Canvas1 op groen en doen hetzelfde met de backgroundcolor van label VAR1. In labelVAR1 staat namelijk de waarde voorgesteld van de variabele die we met de groene grafiek willen voorstellen.





Vervolgens tekenen we een lijn op het canvas. Een lijn wordt altijd getekend tussen twee x,y coördinaten. We moeten het start-coördinaat en het eindpunt-coördinaat opgeven. X1 en Y1 zijn de start-coördinaten. Beiden staan om te starten op 0, maar deze tekeningen gaan er van uit dat we al even in het programma zitten. X1 heeft dus al een bepaalde waarde en Y1 ook. De lijn wordt van het X1,Y1 coördinaat getrokken naar het coördinaat X1+1, wat 1 pixel verder naar rechts ligt (en wat we in deze tekening voor de duidelijkheid heel erg hebben vergroot) en als Y-coördinaat gebruiken we de waarde Value1Out, de eerste van de 3 meetwaarden die we grafisch willen voorstellen. Dit zelfde herhalen we voor de 3 meetwaarden. We geven elke meetwaarde een eigen kleur. Nadien wordt Y1 gelijk gesteld aan Value1Out, Y2 aan Value12Out en Y3 aan Value3Out voordat deze terug worden gevuld met de nieuwe meetwaarden die via BLUETOOTH binnenkomen.



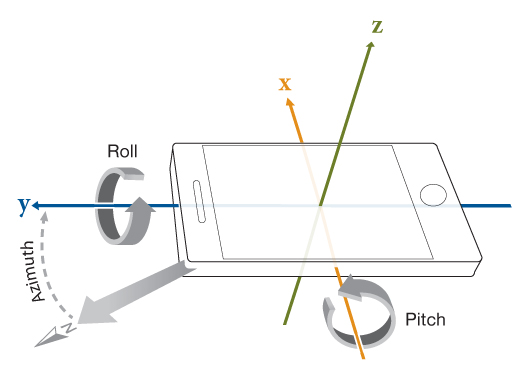
Nu is het X1, Y1 start-coördinaat hetzelfde geworden als wat bij onze vorige stap het eind-coördinaat van de lijn was. Het nieuwe eindpunt wordt nu weerom bepaald door X1+1 en door de nieuwe waarde in Value1Out. Hetzelfde geldt ook voor de 2 andere meetwaarden.

Dit is een manier waarop we grafisch een aantal traag variërende meetwaarden in functie van de tijd kunnen voorstellen. Als u de app uitvoert, dan ziet u hier mooi dat de 3 meetwaarden vertaald worden naar grafieken. Het nadeel van deze methode is dat wanneer het Canvas vol is, de metingen niet meer gevisualiseerd worden. Via een druk op de Clear Button wissen we het canvas, wordt X terug op 0 gezet en kan de visualisering opnieuw starten.

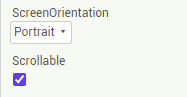
## Sensoren van de smartphone gebruiken

Deze APP stuurt 2 waarden naar de microcontroller. De 2 waarden van de Roll en de van uw SMARTDEVICE.

In de meeste SmartDevices zit er een ‘Orientation’ sensor die detecteert hoe uw SMARTDEVICE gekanteld is. Zo wordt de ‘Roll’-value aangepast als je het SMARTDEVICE kantelt rond de lengte –as en de ‘Pitch’-value wordt aangepast als je het device kantelt rond de breedte-as. Je hebt daarnaast ook nog de ‘Azimuth’ die je de hoekverdraaiing t.o.v. het noorden aangeeft.



De ‘OrientationSensor’ geeft je onder APPINVENTOR 3 mogelijke meetwaarden terug. We bespreken hier even wat de respectievelijke meetwaarden betekenen.

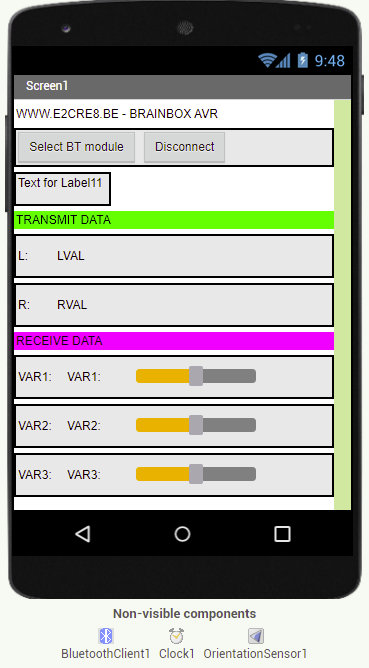
Let wel op: bij de meeste SMARTDEVICES moet je de ‘Automatische Schermrotatie’ uit zetten om de waarde van deze sensor correct uit te kunnen lezen. De ‘Automatische Schermrotatie’ gebruikt deze ‘OrientationSensor’ immers om het scherm te draaien volgens de stand van uw telefoon. De assen van deze sensor worden dan ook mee gedraaid wat ons foutieve meetwaarden zouden kunnen bezorgen.

In meer recente versies van Appinventor moet ook de Screenorientation eigenschap op portrait gezet worden.

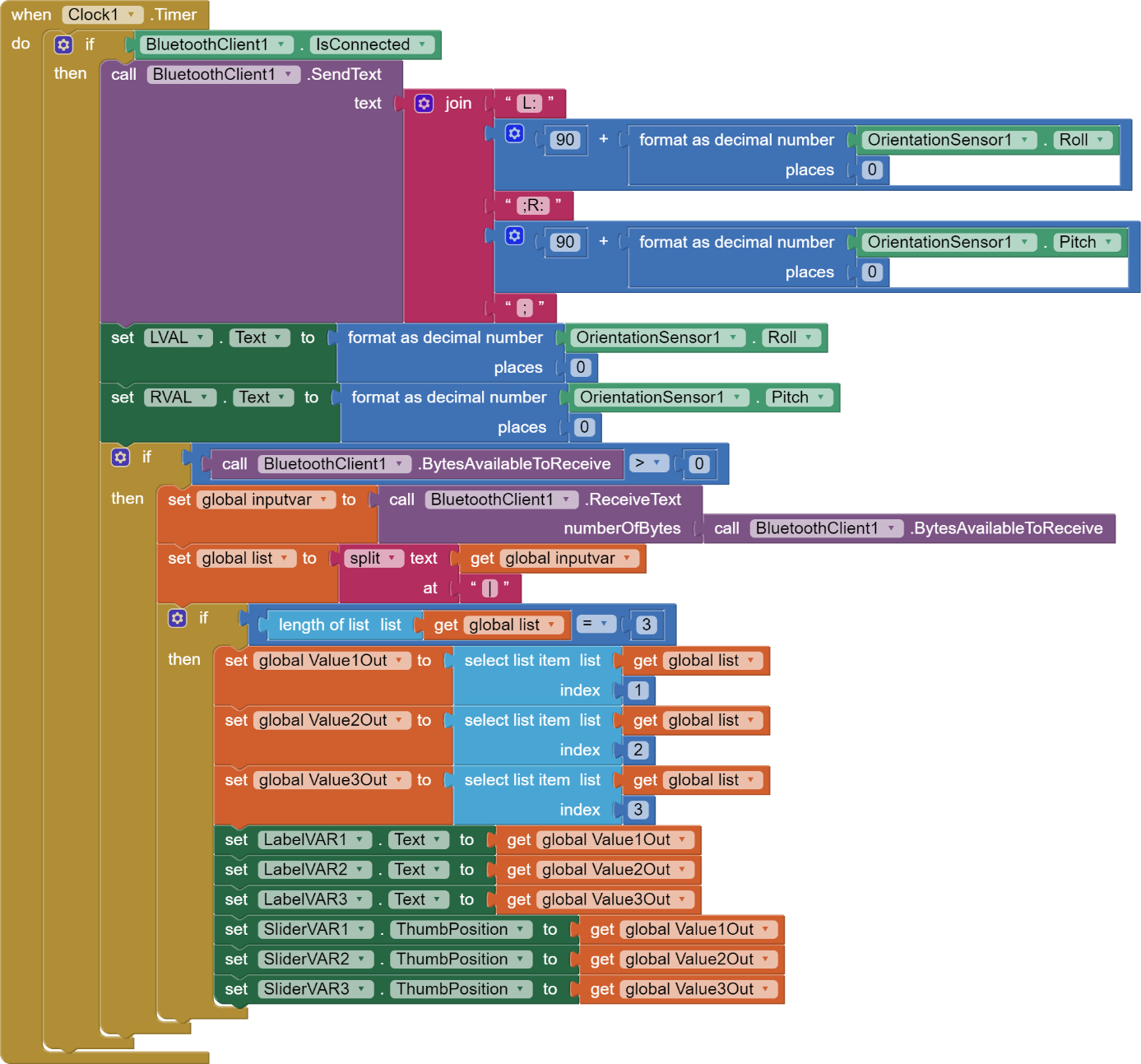
**Roll**: 0° als het device horizontaal ligt, stijgend tot 90° als het device naar links gekanteld wordt en tot -90° als het device naar rechts gekanteld wordt.

**Pitch**: 0° als het device horizontaal ligt, stijgend tot 90° als het device met de bovenkant naar beneden wijst en terug dalend naar 0° als het device nog verder wordt doorgedraaid tot het ondersteboven hangt. Dalend tot -90° als het device met de top naar boven wijst en terug stijgend naar 0° als het verder wordt doorgedraaid tot het ondersteboven hangt.

**Azimuth**: Dit is eigenlijk een kompas. Het geeft een waarde 0 door als het device naar het Noorden gericht wordt. 90° naar het Oosten, 180° naar het Zuiden en 270° naar het Westen.



Hierboven ziet u de opbouw van het scherm in de APPINVENTOR omgeving. Merk op dat dit nagenoeg identiek is aan ons basisprogramma. Enkel de transmit sliders zijn verdwenen.



Je ziet hier dat we aan ons basisprogramma weinig hebben aangepast. Enkel importeren we hier de waarden van de Roll en Pitch sensoren ipv de slider position. We formatteren dit kommagetal naar een geheel getal en tellen er 90 bij om negatieve getallen te vermijden omdat ons Arduino programma daar niet mee om kan.

Voorts verdwijnen enkel de 2 loops die reageren op de verandering in thumbposition van de twee horizontale sliders.

Alle variabelen blijven dezelfde.

