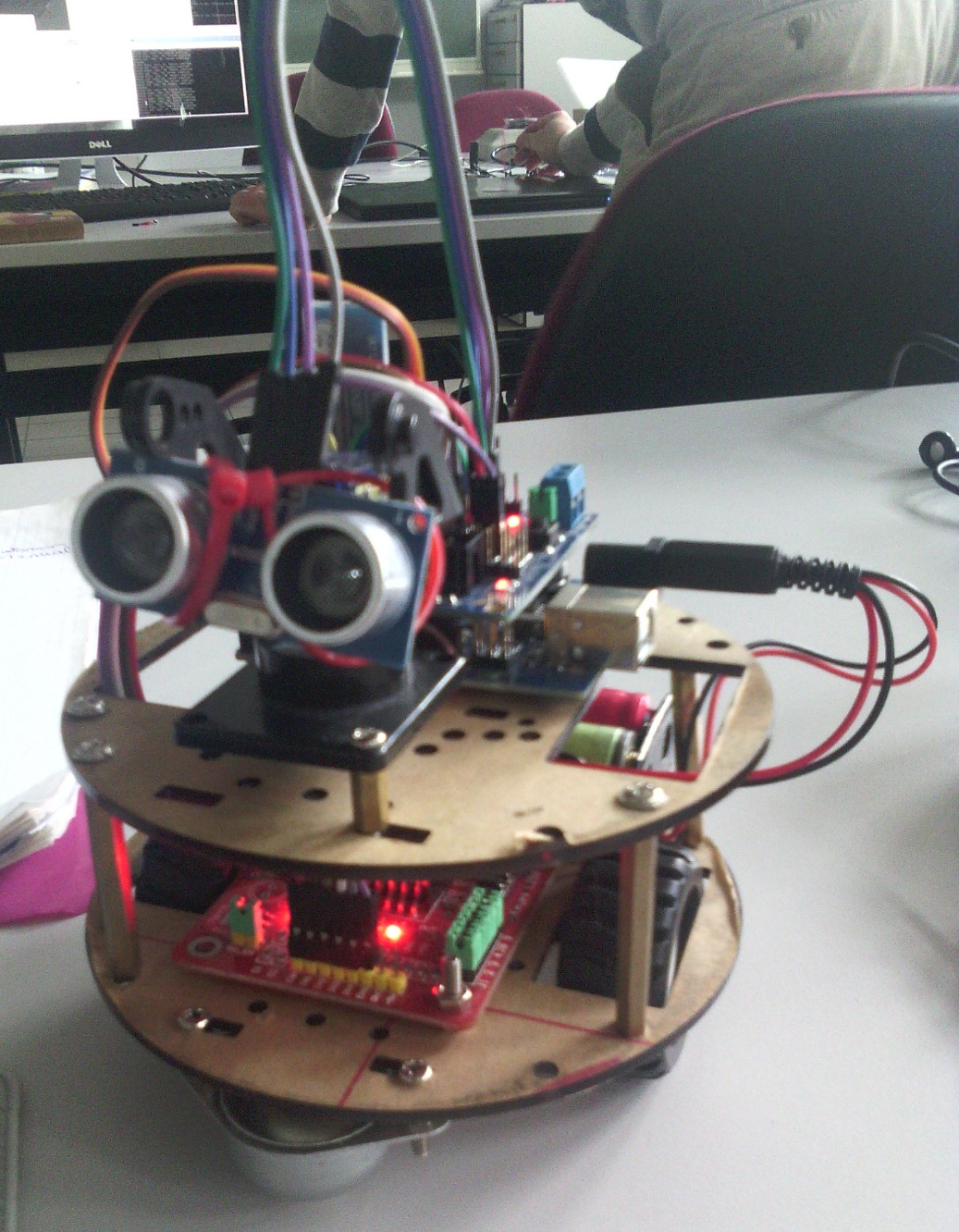
Datacommunication:

Arduino Robot



Bavo De Waele

Hein Pauwelyn

# Inleiding:

Onze opdracht omvat het maken van een robot en deze kunnen besturen d.m.v. een mobiele gsm. Wij besturen de robot met Bluetooth. Naast het besturen van de robot moet deze ook autonoom kunnen rijden op 2 manieren. De 1ste manier is met lichtsensors. Doormiddel van 3 gemonteerde sensoren kan de robot op een zwarte lijn rijden. De sensoren bevinden zich links, rechts en in het midden van het voertuig. De 2de manier is met een sonar en een servo motor.

Wij hebben dit aangepakt door eerst de lijnvolger te maken. Dit was in de 1ste week al afgewerkt. In week 2 werd gewerkt aan de sonar en de servo motor. Week 3 hebben we gewerkt aan de Bluetooth en hem manueel laten rijden. In week 4 hebben wij verder gewerkt aan de applicatie. Doordat er een conflict was in het lesrooster was er niet genoeg tijd om de applicatie volledig te testen. Als samenwerkplaats gebruikten wij GitHub.

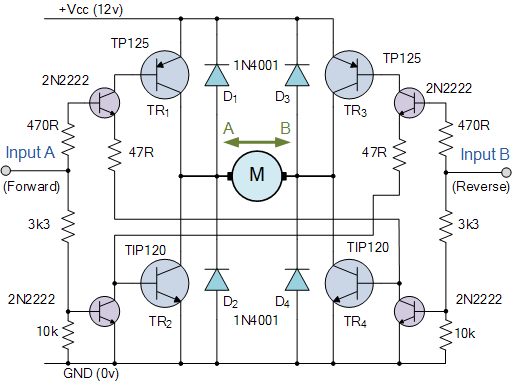
# Lichtsensor:

De sensors worden in een loodrechte positie geplaats van elkaar.

Deze zijn lichtgevoelig. Hierbij registreert de sensor de aanwezigheid of afwezigheid van licht. Bij de registratie van een lichtsignaal wordt dit als 1 weergegeven en bij de afwezigheid van licht is dit 0. Op basis van deze info hebben we de robot geprogrammeerd.

|  |
| --- |
| lichtsensor1 = digitalRead(7);  lichtsensor2 = digitalRead(4);  lichtsensor3 = digitalRead(3);    if(lichtsensor1==HIGH&&lichtsensor3==HIGH){  followingline=true;  backward(1);  }  else if(lichtsensor1==HIGH&&lichtsensor3==LOW){  followingline=true;  turnRight(1);  }  else if(lichtsensor1==LOW&&lichtsensor3==HIGH){  followingline=true;  turnLeft(1);  } |

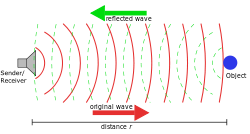
# MOTORS BEDIRECTIONAL SPEED

De motor heeft 2 ingangen. Door de stroom van directie te laten wisselen kan de robot vooruit of achteruit rijden. In dit geval reed de robot vooruit als de stroom van A naar B stroomde. Bij een omgekeerde stroom reed de robot achteruit. De programmatie gebeurde als volgt.

|  |
| --- |
| int speed = 255;  void motorAOn()  {  analogWrite(enableA, speed);  }    void motorBOn()  {  digitalWrite(enableB, speed);  }    //disable motors  void motorAOff()  {  digitalWrite(enableB, LOW);  }    void motorBOff()  {  analogWrite(enableA, LOW);  }  //motor A controls  void motorAForward()  {  digitalWrite(pinA1, HIGH);  digitalWrite(pinA2, LOW);  }    void motorABackward()  {  digitalWrite(pinA1, LOW);  digitalWrite(pinA2, HIGH);  }    //motor B contorls  void motorBForward()  {  digitalWrite(pinB1, LOW);  digitalWrite(pinB2, HIGH);  }    void motorBBackward()  {  digitalWrite(pinB1, HIGH);  digitalWrite(pinB2, LOW);  }    //coasting and braking  void motorACoast()  {  digitalWrite(pinA1, LOW);  digitalWrite(pinA2, LOW);  }    void motorABrake()  {  digitalWrite(pinA1, HIGH);  digitalWrite(pinA2, HIGH);  }    void motorBCoast()  {  digitalWrite(pinB1, LOW);  digitalWrite(pinB2, LOW);  }    void motorBBrake()  {  digitalWrite(pinB1, HIGH);  digitalWrite(pinB2, HIGH);  }    //Define High Level Commands  void enableMotors()  {  motorAOn();  motorBOn();  }    void disableMotors()  {  motorAOff();  motorBOff();  }    void forward(int time)  {  motorAForward();  motorBForward();  delay(time);  }    void backward(int time)  {  motorABackward();  motorBBackward();  delay(time);  }    void turnLeft(int time)  {  motorABackward();  motorBForward();  delay(time);  }    void turnRight(int time)  {  motorAForward();  motorBBackward();  delay(time);  }    void coast(int time)  {  motorACoast();  motorBCoast();  delay(time);  }    void brake(int time)  {  motorABrake();  motorBBrake();  delay(time);  } |

# ULTRA SONAR

De sonar was het moeilijkste deel. Ondanks vele pogingen lukte het ons niet om dit zelf te programmeren. Door middel van een C++ klasse lukte het ons uiteindelijk wel om een juist signaal te ontvangen. Deze ping zorgt ervoor dat de robot nergens tegenaan rijdt. Wij hadden ook een eigen methode ontwikkeld maar deze werkte niet correct en soms toonde hij geen resultaat terwijl er wel 1 moest zijn.

Een sonar werkt d.m.v. een signaal te zenden en deze terug te ontvangen. Je kan dit krijgen door de zenderpin op een hoog signaal te zenden en dan weer laag. Het nadeel van sonar is dat er veel invloed is van andere storende signalen bv een andere sonar of een object die signalen opvangt en ze niet terugzend.

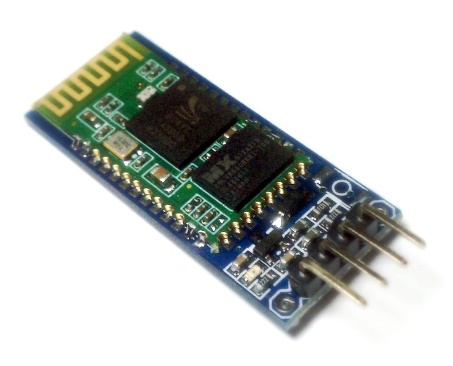
Dit was onze code:

|  |
| --- |
| //SONAR  digitalWrite(trigPin, LOW);  delayMicroseconds(2);  digitalWrite(trigPin, HIGH);  delayMicroseconds(10);    digitalWrite(trigPin, LOW);  duration = pulseIn(echoPin, HIGH);    //Calculate the distance (in cm) based on the speed of sound.  distance = duration/58.2; |

Dit onze code nu:

|  |
| --- |
| #include "NewPing.h"  NewPing sonar(trigPin,echoPin,1500);  //SONAR  distance = sonar.ping\_cm(); |

# BLUETOOTH

Dit is een Bluetooth receiver/sender. Dit werkt doormiddel van 4 pinnen. De 2 buitenste zijn de power pinnen. De 2 binnenste zijn de RX en TX. Bluetooth zelf verbruikt niet veel en kan over korte afstand gestuurd worden. Dit was dan ook ideaal voor deze opdracht. Bluetooth communiceert via COM poorten wat het dus vrij makkelijk maakt om te programmeren via de arduino kant. Dit is gewoon via serial klasse sturen.

Dit is hoe wij de Bluetooth geprogrammeerd hebben langs de arduino kant.

|  |
| --- |
| int ibyte =Serial.read();    if(ibyte=='m' ||ibyte=='M')  {  autonoom=false;  }  if(ibyte=='o'||ibyte=='O')  {  autonoom=true;  } |

Naast de Bluetooth kant van arduino hebben we ook de Android kant. Dit was iets ingewikkelder. Dit is het gevolgde stappenplan.

1. Kijk of het device een Bluetooth zender/receiver heeft.
2. Kijk of de Bluetooth aanstaat
   1. Zo niet, zet hem aan
3. Haal alle connected devices op en toon ze in een lijst.
4. Vanaf dat er een device geselecteerd is word deze geconnecteerd
5. Als er connectie is, kan communicatie beginnen.

Dit is het 1ste deel. Deze controleert of er een Bluetooth adapter is. Daarna checkt hij de gelinkte devices en voegt deze toe aan een adapter.

|  |
| --- |
| **myBluetooth** = BluetoothAdapter.*getDefaultAdapter*();  **if**(**myBluetooth** == **null**){  Toast.*makeText*(getApplicationContext(),**"BLUETHOOTH NOT SUPPORTED"**,Toast.***LENGTH\_SHORT***).show();  finish(); } **else**{  **if**(!**myBluetooth**.isEnabled()){  Intent intent = **new** Intent(BluetoothAdapter.***ACTION\_REQUEST\_ENABLE***);  startActivityForResult(intent,1);  } }  **bluetoothDevices** = **myBluetooth**.getBondedDevices();  DeviceAdapter bluetoothDeviceArrayAdapter = **new** DeviceAdapter(**this**, android.R.layout.***simple\_spinner\_dropdown\_item***);  **if**(**bluetoothDevices**.size()!=0){   **for**(BluetoothDevice device:**bluetoothDevices**){  bluetoothDeviceArrayAdapter.add(device);  Log.*e*(**"Bluethooth device"**,device.toString());  } |

Hierna word deze adapter geplaatst in een listview. Als er 1 geselecteerd word, schieten deze 2 classes in gang. De 1ste connecteert en eenmaal er connectie is zorgt de 2de klasse voor het ontvangen en zenden over het Bluetooth netwerk.

|  |
| --- |
| **public class** ConnectThread **extends** Thread {  **private final** BluetoothSocket **mmSocket**;  **private final** BluetoothDevice **mmDevice**;  **private final** UUID **MY\_UUID**;  **private final** BluetoothAdapter **mBluetoothAdapter**;  **private final** ConnectThreadListener **listener**;  **public** ConnectThread(ConnectThreadListener listener, BluetoothDevice device,UUID MY\_UUID,BluetoothAdapter mBluetoothAdapter) {  *// Use a temporary object that is later assigned to mmSocket,  // because mmSocket is final* BluetoothSocket tmp = **null**;  **mmDevice** = device;   **this**.**MY\_UUID** = MY\_UUID;  **this**.**mBluetoothAdapter** = mBluetoothAdapter;  **this**.**listener** = listener;   *// Get a BluetoothSocket to connect with the given BluetoothDevice* **try** {  *// MY\_UUID is the app's UUID string, also used by the server code* tmp = device.createInsecureRfcommSocketToServiceRecord(MY\_UUID);  } **catch** (IOException e) { }  **mmSocket** = tmp;  }   **public void** run() {  *// Cancel discovery because it will slow down the connection  //mBluetoothAdapter.cancelDiscovery();* **try** {  *// Connect the device through the socket. This will block* **mBluetoothAdapter**.cancelDiscovery();  *// until it succeeds or throws an exception* **mmSocket**.connect();   } **catch** (IOException connectException) {  *// Unable to connect; close the socket and get out* Log.*e*(**"Status"**, **"Could not connect!"**);  **listener**.CouldNotConnectToSocket();  **try** {  **mmSocket**.close();   } **catch** (IOException closeException) {   Log.*e*(**"Status"**, **"Could not close!"**);  }  *//return;* }  Log.*e*(**"Status"**, **"Connected"**);   *// Do work to manage the connection (in a separate thread)* **listener**.manageConnectedSocket(**mmSocket**);  }   */\*\* Will cancel an in-progress connection, and close the socket \*/* **public void** cancel() {  **try** {  **mmSocket**.close();  } **catch** (IOException e) { }  }   **public interface** ConnectThreadListener{  **public void** manageConnectedSocket(BluetoothSocket mmSocket);  **public void** CouldNotConnectToSocket();  } }  **public class** ConnectedThread **extends** Thread {  **private final** BluetoothSocket **mmSocket**;  **private final** InputStream **mmInStream**;  **private final** OutputStream **mmOutStream**;   **public** ConnectedThread(BluetoothSocket socket) {  **mmSocket** = socket;  InputStream tmpIn = **null**;  OutputStream tmpOut = **null**;   *// Get the input and output streams, using temp objects because  // member streams are final* **try** {  tmpIn = socket.getInputStream();  tmpOut = socket.getOutputStream();  } **catch** (IOException e) { }   **mmInStream** = tmpIn;  **mmOutStream** = tmpOut;  }   **public void** run() {  **byte**[] buffer = **new byte**[1024]; *// buffer store for the stream* **int** bytes; *// bytes returned from read()   // Keep listening to the InputStream until an exception occurs* **while** (**true**) {  **try** {  *// Read from the InputStream* bytes = **mmInStream**.read(buffer);  *// Send the obtained bytes to the UI activity  //mHandler.obtainMessage(MESSAGE\_READ, bytes, -1, buffer)  // .sendToTarget();* } **catch** (IOException e) {  **break**;  }  }  }   */\* Call this from the main activity to send data to the remote device \*/* **public void** write(**byte**[] bytes) {  **try** {  **mmOutStream**.write(bytes);  } **catch** (IOException e) { }  }   */\* Call this from the main activity to shutdown the connection \*/* **public void** cancel() {  **try** {  **mmSocket**.close();  } **catch** (IOException e) { }  } } |

# SLOT

Tijdens deze opdracht hebben wij veel bijgeleerd. Om te beginnen was dit de eerste keer dat we in groep moesten coderen. Dit is natuurlijk geen makkelijke opdracht. Wij zijn er wel in geslaagd om dit zo goed mogelijk te doen. Daarnaast hebben wij ook veel technische dingen bijgeleerd. Het is toch een uitdaging om een robot zo te coderen dat hij een lijn volgt. De codes die we hebben geleerd, zullen we later ook kunnen gebruiken. Voor toekomstige studenten zouden wij niets veranderen. Wij vonden elk aspect van deze opdracht goed en leerzaam.