Maak van de volgende tekst een duidelijke educatieve beschrijving op een zo’n hoog mogelijk niveau.

**Visual Studio C++ Tutorial GUI servo interface**

**Geavanceerde Uitleg: Visual Studio C++ GUI Servo Interface**

**Inleiding**

In deze tutorial ontwerpen en implementeren we een grafische gebruikersinterface (GUI) in Visual Studio C++ om servo's te besturen via een Arduino. Met deze interface kunnen gebruikers servo's eenvoudig bedienen, waarden instellen via sliders en complexe sequences opslaan en afspelen. Dit project combineert kennis van servo-aansturing via een timer-interrupt op de Arduino met geavanceerde GUI-programmering en serial communicatie.

Om de ontwikkeling beheersbaar en gestructureerd te houden, hanteren we een iteratieve aanpak: **begin klein en breid later uit**. Elk onderdeel wordt in afzonderlijke stappen uitgewerkt.

**Specificaties van de GUI**

De GUI moet in eerste instantie de volgende functionaliteiten bevatten:

1. **Verbinding maken met een Arduino via een seriële poort.**
2. **Sliders om servo-posities in te stellen.**
3. **De mogelijkheid om waardes naar de Arduino te verzenden en bevestigingen te ontvangen.**
4. **Opslag van slider-waarden in een sequence.**
5. **Afspelen van sequences door serial commands te verzenden en de Arduino deze te laten uitvoeren.**

**Stap-voor-stap Aanpak**

**Stap 1: Verbinding maken met een Arduino (UART)**  
Doel: Een stabiele seriële verbinding tot stand brengen tussen de GUI en de Arduino.

* Implementeer een seriële communicatieklasse die een COM-poort opent, sluit en beheert.
* Voeg foutafhandeling toe om verbindingsproblemen, zoals een niet-beschikbare poort, correct af te handelen.

**Stap 2: Serial Data Verzenden**  
Doel: Stuur data van de GUI naar de Arduino via UART.

* Creëer functies om slider-waarden om te zetten naar seriële gegevens (bijvoorbeeld als een string in het formaat "<slider1>:<value1>").
* Zorg dat de Arduino commando's correct kan interpreteren.

**Stap 3: Serial Data Ontvangen**  
Doel: Ontvang bevestigingen of feedback van de Arduino.

* Implementeer een event-gedreven aanpak die gegevens leest zodra de Arduino data verzendt.
* Toon ontvangen gegevens in de GUI, bijvoorbeeld in een **TextBox** of console-uitvoer.

**Stap 4: Variabelen Veranderen via Sliders**  
Doel: Gebruik sliders in de GUI om real-time wijzigingen aan te brengen in servo-posities.

* Stel sliders in met een bereik van minimaal 500 µs tot maximaal 2500 µs, de standaard servo-pulsebreedte.
* Voeg een event-handler toe die de actuele waarde van de slider weergeeft in een gekoppelde **TextBox**.

**Stap 5: Slider Waarden Verzenden**  
Doel: Stuur slider-waarden direct naar de Arduino wanneer ze worden gewijzigd.

* Implementeer functies die bij iedere slider-update een nieuwe waarde naar de Arduino verzenden.

**Stap 6: Slider Waarden Tonen**  
Doel: Toon de slider-waarden zoals ontvangen door de Arduino in de GUI.

* Zorg dat feedback van de Arduino op het scherm verschijnt, zodat je kunt bevestigen dat de juiste gegevens worden ontvangen.

**Stap 7: Waarden Opslaan in een Sequence**  
Doel: Voeg meerdere sets van slider-waarden op in een array of vector in de GUI.

* Bij het klikken op een **Save**-knop worden de huidige slider-waarden opgeslagen in een dynamische lijst (bijvoorbeeld een std::vector).
* Toon opgeslagen sequences in een **ListBox** of **TextField**.

**Stap 8: Sequences Afspelen**  
Doel: Stuur de opgeslagen sequences naar de Arduino en speel deze af.

* Voeg een **Play**-knop toe die de gehele sequence naar de Arduino stuurt.
* De Arduino verwerkt de sequence en beweegt de servo's overeenkomstig.

**Overzicht**

Je kunt dan een overzicht maken je ziet dan in grote lijnen wat elk onderdeel moet doen.

**GUI**

arduino

UART

Serial data versturen

Position variable

slider

tonen

Speed variable

text

text

Save position sequence

play Movement

next

save

textFile

Load

Dit zal de eerste keer niet gelijk helemaal goed zijn.

Value max

times

value

timer

value

Slider

Value min

**Arduino**

UART

Arduino

Serial data ontvangen

Serial data play

Serial data Read SD

Serial data save SD

upload Patern

Position variable

play Movement

run

Speed variable

times

type

next

Position, speed

CPG

Timer interupt

servo

**LAYOUT**

BAUD

COM

roll

pitch

Installeren visual studio C++/CLI

Starten new project.

properties

Instellen form main

Project naam

Layout Form

Layout form 2

**UART**

**Verbinden met com port**

private: System::Void MyForm\_Load(System::Object^ sender, System::EventArgs^ e) {

array<Object^>^ comport = SerialPort::GetPortNames();

cbPort->Items->AddRange(comport);

cbPort->SelectedIndex = 0;

**Instellen baud rate**

array<Object^>^ baudrate = { 9600, 57600, 115200 };

cbBaud->Items->AddRange(baudrate);

cbBaud->SelectedIndex = 0;

connect

btnConnect

**Ontvangen serial data buffer to textfield**

txtBox1 multiline

private: System::Void sp\_DataReceived(System::Object^ sender, System::IO::Ports::SerialDataReceivedEventArgs^ e) {

// Controleer of de seriële poort nog steeds open is

if (sp->IsOpen) {

try {

// Lees de seriële gegevens

String^ data = sp->ReadLine() + "\n";

textBox1->Invoke(gcnew Action<String^>(this, &ArduinoGui::AppendTextToTextBox), data);

}

catch (IOException^ ex) {

MessageBox::Show("Error reading from serial port: " + ex->Message);

}

}

else {

MessageBox::Show("Serial port is closed.");

}

}

**Verzenden serial data -> textbox2 -> btnSend**

textBox2

send

btnSend

private: System::Void btnSend\_Click(System::Object^ sender, System::EventArgs^ e) {

String^ message = this->textBox2->Text;

if (this->sp->IsOpen)

{

this->sp->WriteLine(message);

}

}

**Verzenden slider values -> show values -> textbox**

hScrollBar1

lbl1

//===================== Form::Timer ======================================

private:

System::Windows::Forms::Timer^ updateTimer; // Timer om updates te regelen

int pendingValue1 = -1; // Buffer voor hScrollBar1

int pendingValue2 = -1; // Buffer voor hScrollBar2

int pendingValue3 = -1; // Buffer voor hScrollBar3

int lastSentValue1 = -1; // Laatst verzonden waarde van hScrollBar1

int lastSentValue2 = -1; // Laatst verzonden waarde van hScrollBar2

int lastSentValue3 = -1; // Laatst verzonden waarde van hScrollBar3

// Initialisatie van de timer (uit te voeren in de constructor of init-functie)

void InitializeUpdateTimer() {

updateTimer = gcnew System::Windows::Forms::Timer();

updateTimer->Interval = 100; // Interval van 100 ms

updateTimer->Tick += gcnew System::EventHandler(this, &ArduinoGui::OnUpdateTimerTick);

updateTimer->Start(); // Start de timer

}

// Timer callback om de waarden van de scrollbars te verwerken

void OnUpdateTimerTick(System::Object^ sender, System::EventArgs^ e) {

try {

// Verwerk hScrollBar1

if (pendingValue1 != -1 && System::Math::Abs(lastSentValue1 - pendingValue1) >= 10) {

String^ outputText1 = "S1:" + pendingValue1.ToString();

sp->WriteLine(outputText1);

Console::WriteLine("Verzonden naar servo 1: " + outputText1);

lastSentValue1 = pendingValue1;

pendingValue1 = -1; // Reset buffer

}

// Verwerk hScrollBar2

if (pendingValue2 != -1 && System::Math::Abs(lastSentValue2 - pendingValue2) >= 10) {

String^ outputText2 = "S2:" + pendingValue2.ToString();

sp->WriteLine(outputText2);

Console::WriteLine("Verzonden naar servo 2: " + outputText2);

lastSentValue2 = pendingValue2;

pendingValue2 = -1; // Reset buffer

}

// Verwerk hScrollBar3

if (pendingValue3 != -1 && System::Math::Abs(lastSentValue3 - pendingValue3) >= 10) {

String^ outputText3 = "S3:" + pendingValue3.ToString();

sp->WriteLine(outputText3);

Console::WriteLine("Verzonden naar servo 3: " + outputText3);

lastSentValue3 = pendingValue3;

pendingValue3 = -1; // Reset buffer

}

}

catch (Exception^ ex) {

Console::WriteLine("Fout tijdens seriële communicatie: " + ex->Message);

}

}

// Scrollbar ValueChanged-events

System::Void hScrollBar1\_ValueChanged(System::Object^ sender, System::EventArgs^ e) {

UpdateScrollbar(hScrollBar1, lbl11, pendingValue1);

}

System::Void hScrollBar2\_ValueChanged(System::Object^ sender, System::EventArgs^ e) {

UpdateScrollbar(hScrollBar2, lbl7, pendingValue2);

}

System::Void hScrollBar3\_ValueChanged(System::Object^ sender, System::EventArgs^ e) {

UpdateScrollbar(hScrollBar3, lbl8, pendingValue3);

}

// Hulpfunctie om de waarde van een scrollbar te verwerken

void UpdateScrollbar(System::Windows::Forms::HScrollBar^ scrollbar, System::Windows::Forms::Label^ label, int% pendingValue) {

// Stel de minimum en maximum waarden in

scrollbar->Minimum = 0;

scrollbar->Maximum = 180;

// Zorg ervoor dat de waarde binnen de geldige grenzen blijft

if (scrollbar->Value < scrollbar->Minimum) {

scrollbar->Value = scrollbar->Minimum;

}

else if (scrollbar->Value > scrollbar->Maximum) {

scrollbar->Value = scrollbar->Maximum;

}

// Werk de waarde in de label bij

label->Text = scrollbar->Value.ToString();

// Sla de waarde op in de buffer voor verwerking door de timer

pendingValue = scrollbar->Value;

}

//=====================================================================================

**Trouble we moeten de serialPort tijd genoeg geven om de values te versturen.**

**Save slider values -> btnSave -> txtBox3**

**next**

//=================== save sequences ===============================

// Lijst om de sequenties op te slaan

private: System::Collections::Generic::List<String^>^ sequences = gcnew System::Collections::Generic::List<String^>();

// Initialiseer een teller voor de sequentienamen

private: int sequenceCounter = 1;

// btnSave Click-event

private: System::Void btnSave\_Click\_1(System::Object^ sender, System::EventArgs^ e) {

try {

// Controleer of labels waarden bevatten

if (String::IsNullOrEmpty(lbl11->Text) || String::IsNullOrEmpty(lbl7->Text) || String::IsNullOrEmpty(lbl8->Text)) {

MessageBox::Show("Een of meer labels hebben geen waarde!", "Fout");

return;

}

// Haal de huidige waarden van de labels

String^ currentSequence = "Sequence" + sequenceCounter.ToString() + ": ";

currentSequence += "S1=" + lbl11->Text + ", ";

currentSequence += "S2=" + lbl7->Text + ", ";

currentSequence += "S3=" + lbl8->Text;

// Debug: Controleer of de string is opgebouwd

Console::WriteLine("Opgebouwde sequentie: " + currentSequence);

// Voeg de nieuwe sequentie toe aan de lijst

sequences->Add(currentSequence);

// Update de TextBox met de opgeslagen sequenties

textSequence->AppendText(currentSequence + Environment::NewLine);

// Debug: Controleer of de TextBox is bijgewerkt

Console::WriteLine("TextBox bijgewerkt met: " + currentSequence);

// Verhoog de sequentieteller

sequenceCounter++;

}

catch (Exception^ ex) {

MessageBox::Show("Fout bij opslaan van de sequentie: " + ex->Message, "Fout");

}

}

//====================================================================

**play movement -> send movement**

**Trouble we moeten de serialPort tijd genoeg geven om de values te versturen.**

//=========================== Play Sequences =================================

private:

bool isPlaying = false; // Vlag om aan te geven of de sequences worden afgespeeld

int currentSequenceIndex = 0; // Huidige index van de sequence

System::Void btnPlay\_Click(System::Object^ sender, System::EventArgs^ e) {

try {

// Zet de vlag op true om afspelen te starten

isPlaying = true;

currentSequenceIndex = 0; // Start bij de eerste sequence

// Start de afspeellogica in een aparte thread

System::Threading::Thread^ playThread = gcnew System::Threading::Thread(gcnew System::Threading::ThreadStart(this, &ArduinoGui::PlaySequences));

playThread->Start();

}

catch (Exception^ ex) {

MessageBox::Show("Fout tijdens het afspelen: " + ex->Message, "Fout");

}

}

void PlaySequences() {

// Loop door de sequences in de achtergrondthread

while (isPlaying && currentSequenceIndex < sequences->Count) {

String^ sequence = sequences[currentSequenceIndex];

// Verdeel de sequence in de verschillende servo-posities

array<String^>^ parts = sequence->Split(',');

if (parts->Length < 3) continue;

// Extract de servo posities

int s1Pos = Convert::ToInt32(parts[0]->Split('=')[1]);

int s2Pos = Convert::ToInt32(parts[1]->Split('=')[1]);

int s3Pos = Convert::ToInt32(parts[2]->Split('=')[1]);

// Stuur alle servo posities tegelijk naar de Arduino

sp->WriteLine("S1:" + s1Pos);

sp->WriteLine("S2:" + s2Pos);

sp->WriteLine("S3:" + s3Pos);

System::Threading::Thread::Sleep(500);

currentSequenceIndex++;

if (currentSequenceIndex == sequences->Count) {

currentSequenceIndex = 0;

}

}

MessageBox::Show("Alle sequences zijn succesvol afgespeeld!", "Informatie");

}

//=============== stop playing sequences =====================================

System::Void btnStop\_Click(System::Object^ sender, System::EventArgs^ e) {

// Zet de vlag op false om de afspeellijst te stoppen

isPlaying = false;

MessageBox::Show("Het afspelen van de sequences is gestopt.", "Informatie");

}

//=============================================================================

Sequence save

Textbox en label waardes worden in de array geplaatst en naar de textbox verzonden.

Play sequence de array wordt afgespeeld

De tekst uit de tekst box moet in een array komen.

En deze moet worden afgespeelt

Tekst vanuit een file moet in de tekst box komen

//=========================== Play Sequences ==================================================================

private:

bool isPlaying = false; // Vlag om aan te geven of de sequences worden afgespeeld

int currentSequenceIndex = 0; // Huidige index van de sequence

private: System::Void btnPlay\_Click(System::Object^ sender, System::EventArgs^ e) {

try {

// Zet de vlag op true om afspelen te starten

isPlaying = true;

currentSequenceIndex = 0; // Start bij de eerste sequence

Console::WriteLine("Ok btnPlay clicked: ");

// Start de afspeellogica in een aparte thread via BackgroundWorker

BackgroundWorker^ playWorker = gcnew BackgroundWorker();

playWorker->DoWork += gcnew DoWorkEventHandler(this, &ArduinoGui::PlaySequences);

playWorker->RunWorkerCompleted += gcnew RunWorkerCompletedEventHandler(this, &ArduinoGui::PlaySequencesCompleted);

playWorker->RunWorkerAsync(); // Start de achtergrondtaak

}

catch (Exception^ ex) {

MessageBox::Show("Fout tijdens het afspelen: " + ex->Message, "Fout");

}

}

void PlaySequences(Object^ sender, DoWorkEventArgs^ e) {

Console::WriteLine("Aantal sequences: " + sequences->Count);

if (sequences->Count == 0) {

Console::WriteLine("Geen sequences beschikbaar.");

isPlaying = false;

return;

}

while (isPlaying && currentSequenceIndex < sequences->Count) {

Console::WriteLine("while PlaySequence: " + isPlaying);

try {

String^ sequence = sequences[currentSequenceIndex];

//====================================================

Console::WriteLine("Current Sequence: " + sequence);

// Verwijder de naam van de sequentie, bijvoorbeeld 'Sequence2:'

int colonIndex = sequence->IndexOf(':');

if (colonIndex != -1) {

sequence = sequence->Substring(colonIndex + 1); // Pak alles na ':'

sequence = sequence->Trim(); // Verwijder extra spaties

}

array<String^>^ servoCommands = sequence->Split(';'); // Split de sequentie op ';'

for each (String ^ command in servoCommands) {

command = command->Trim(); // Verwijder spaties

if (command->StartsWith("S1=")) {

String^ position1 = command->Substring(3, command->IndexOf(",") - 3);

String^ speed1 = command->Substring(command->IndexOf("Speed=") + 6);

String^ output1 = "S1:" + position1 + ",Speed=" + speed1;

Console::WriteLine("Processing Command: " + output1);

if (sp->IsOpen) {

sp->WriteLine(output1);

Console::WriteLine("Verzonden naar servo 1: " + output1);

}

else {

Console::WriteLine("Seriële poort niet open voor S1.");

}

}

else if (command->StartsWith("S2=")) {

String^ position2 = command->Substring(3, command->IndexOf(",") - 3);

String^ speed2 = command->Substring(command->IndexOf("Speed=") + 6);

String^ output2 = "S2:" + position2 + ",Speed=" + speed2;

Console::WriteLine("Processing Command: " + output2);

if (sp->IsOpen) {

sp->WriteLine(output2);

Console::WriteLine("Verzonden naar servo 2: " + output2);

}

else {

Console::WriteLine("Seriële poort niet open voor S2.");

}

}

else if (command->StartsWith("S3=")) {

String^ position3 = command->Substring(3, command->IndexOf(",") - 3);

String^ speed3 = command->Substring(command->IndexOf("Speed=") + 6);

String^ output3 = "S3:" + position3 + ",Speed=" + speed3;

Console::WriteLine("Processing Command: " + output3);

if (sp->IsOpen) {

sp->WriteLine(output3);

Console::WriteLine("Verzonden naar servo 3: " + output3);

}

else {

Console::WriteLine("Seriële poort niet open voor S3.");

}

}

else {

Console::WriteLine("Onbekend commando: " + command);

}

}

int wacht = 1000;

wacht = Convert::ToInt32(txtDelay->Text);

System::Threading::Thread::Sleep(wacht);

currentSequenceIndex++;

if (currentSequenceIndex == sequences->Count) {

Console::WriteLine("Einde van sequences bereikt, herstarten.");

currentSequenceIndex = 0; // Optioneel: herstarten van de loop

}

}

catch (Exception^ ex) {

Console::WriteLine("Fout tijdens het afspelen: " + ex->Message);

isPlaying = false; // Stop bij een fout

}

}

isPlaying = false;

Console::WriteLine("Alle sequences zijn succesvol afgespeeld!");

}

void PlaySequencesCompleted(Object^ sender, RunWorkerCompletedEventArgs^ e) {

isPlaying = false;

MessageBox::Show("Alle sequences zijn succesvol afgespeeld!", "Informatie");

}

//==================== eind vervanging ==================================

private: System::Void btnSaveFile\_Click(System::Object^ sender, System::EventArgs^ e) {

SaveFileDialog^ saveFileDialog = gcnew SaveFileDialog();

saveFileDialog->Filter = "Text Files|\*.txt";

saveFileDialog->Title = "Opslaan van sequences";

if (saveFileDialog->ShowDialog() == System::Windows::Forms::DialogResult::OK) {

try {

System::IO::File::WriteAllText(saveFileDialog->FileName, textSequence->Text);

MessageBox::Show("Sequences succesvol opgeslagen!", "Succes", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

catch (Exception^ ex) {

MessageBox::Show("Fout tijdens het opslaan: " + ex->Message, "Fout", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

}

private: System::Void btnLoadFile\_Click(System::Object^ sender, System::EventArgs^ e) {

OpenFileDialog^ openFileDialog = gcnew OpenFileDialog();

openFileDialog->Filter = "Text Files|\*.txt";

openFileDialog->Title = "Laden van sequences";

if (openFileDialog->ShowDialog() == System::Windows::Forms::DialogResult::OK) {

try {

String^ fileContent = System::IO::File::ReadAllText(openFileDialog->FileName);

textSequence->Text = fileContent;

MessageBox::Show("Sequences succesvol geladen!", "Succes", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

catch (Exception^ ex) {

MessageBox::Show("Fout tijdens het laden: " + ex->Message, "Fout", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

}

Probleem sequence opslaan op de arduino

Weinig ruimte of via flash

Opslaan op sd card en de sequence vanaf hier lezen en uitvoeren.

**Stap-voor-stap: Naar geavanceerde servo-aansturing**

**1. Verkennen van seriële communicatie**

* **Doelstelling:** Het fundament leggen door te leren hoe data via seriële communicatie naar de Arduino kan worden verzonden en ontvangen.
* **Resultaat:** Een solide communicatiestructuur waarmee we servo’s kunnen aansturen door gegevens te interpreteren en correct door te sturen naar de hardware.

**2. Implementatie van real-time bediening via een slider**

* **Doelstelling:** De servo’s direct laten reageren op een sliderinterface, zodat de gebruiker posities in real-time kan wijzigen.
* **Kritieke aspecten:** Het waarborgen van voldoende tijd voor de servo om zijn doelpositie te bereiken zonder dat commando’s elkaar overlappen of interfereren.
* **Resultaat:** Een vloeiende bediening waarbij de servo's naadloos de gewenste beweging volgen.

**3. Posities vastleggen en herhaald afspelen**

* **Doelstelling:** Meerdere posities opslaan en een reeks bewegingen ontwerpen die automatisch kunnen worden afgespeeld.
* **Proces:**
  + Posities instellen via de slider en deze in een sequentie opslaan.
  + Meerdere posities combineren tot een volledige beweging.
  + De opgeslagen sequentie in een herhaalbare lus afspelen.
* **Resultaat:** Efficiënt beheer van complexe bewegingspatronen.

**4. Configureren van snelheid en asynchrone beweging**

* **Doelstelling:** Servo’s met individuele snelheden en onafhankelijke bewegingen laten opereren.
* **Toepassing:** Dit opent de deur naar dynamische toepassingen zoals coördinatie tussen meerdere servo’s met verschillende snelheden en timing.
* **Resultaat:** Een veelzijdige en flexibele besturing voor geavanceerde bewegingen.

**5. Persistentie: Posities opslaan en bewerken**

* **Doelstelling:** Servo-posities opslaan in tekstbestanden voor toekomstige referentie of aanpassing.
* **Proces:** De gebruiker kan opgeslagen configuraties eenvoudig aanpassen en opnieuw toepassen.
* **Resultaat:** Vereenvoudigde iteratie en langdurige opslag van instellingen.

**6. Laden en afspelen van opgeslagen bewegingen**

* **Doelstelling:** Opslagbestanden gebruiken om eerder ontworpen bewegingen af te spelen zonder herprogrammering.
* **Resultaat:** Directe toegang tot vooraf geconfigureerde bewegingen, wat zowel tijd als moeite bespaart.

**7. Visualiseren van servo-bewegingen**

* **Doelstelling:** De bewegingen van de servo’s grafisch weergeven om een diepgaander inzicht te verkrijgen in hun dynamiek.
* **Proces:** Het plotten van posities in een grafiek, waarbij trends en patronen zichtbaar worden.
* **Resultaat:** Een krachtig hulpmiddel voor analyse en optimalisatie van bewegingen.

**8. Introduceren van sinusgolven**

* **Doelstelling:** Bewegingen modelleren met sinusvormige golven en deze parametriseren met variabelen zoals fase, frequentie, amplitude en offset.
* **Toepassing:** Dit maakt vloeiende en herhaalbare bewegingen mogelijk die dichter bij natuurlijke patronen liggen.
* **Resultaat:** Servo-bewegingen die veel realistischer en esthetischer zijn.

**9. Servo-aansturing via sinusgolven**

* **Doelstelling:** Het toepassen van sinusgolven als directe invoer voor servo-bewegingen.
* **Resultaat:** Continuïteit en harmonie in de bewegingsuitvoering, ideaal voor complexe robotica-toepassingen.

**10. Ontwikkeling van een Central Pattern Generator (CPG)**

* **Doelstelling:** Een neuraal netwerk ontwerpen dat een geavanceerd bewegingspatroon genereert door gebruik te maken van CPG-technieken.
* **Toepassing:** Ideaal voor het simuleren van biologische bewegingen zoals wandelen, zwemmen of golvende bewegingen.
* **Resultaat:** Een robuust algoritme dat nauwkeurige en aanpasbare bewegingen mogelijk maakt.

**11. Analyse en verfijning van snelheidseffecten**

* **Doelstelling:** De impact van snelheidsvariaties op de servo-bewegingen onderzoeken en fine-tunen.
* **Toepassing:** Het balanceren van snelheid en nauwkeurigheid om optimale prestaties te bereiken.
* **Resultaat:** Een beter begrip van servo-dynamiek en de mogelijkheid om bewegingen verder te optimaliseren.

Maak een neural netwerk met een central pattern generator, voor 3 servo’s

Hierbij zal de ideale werking van het geheel worden vastgelegd door het verschil van elke servo

In amplitude, frequentie, phase en offset. Toon hoe ik het neural netwerk moet opstellen om dit in te kunnen stellen naar het juiste pattern.

Wat we al weten is dat bij het verhogen van de snelheid de yaw beweging kleiner wordt.

Verhogen frequentie roll -> frequentie pitch - > frequentie yaw

Amplitude - > amplitude - > amplitude

Phase - > Phase - > phase

Offset - > Offset - > offset

Een **Central Pattern Generator (CPG)** voor een neurale netwerkstructuur die servo’s aanstuurt, kan worden gerealiseerd met een oscillatormodel dat sinusgolven genereert. De kern is om per servo variabelen als amplitude, frequentie, fase en offset te controleren. Dit model kan in Python worden geïmplementeerd met behulp van een eenvoudige feedforward-netwerkarchitectuur.

Hier is een stapsgewijze aanpak:

**1. Begrippen en doelen**

* **Amplitude (AAA)**: Regelt de bewegingsuitslag van de servo.
* **Frequentie (fff)**: Bepaalt de snelheid van de oscillatie.
* **Fase (ϕ\phiϕ)**: Controleert de relatieve timing tussen servo’s.
* **Offset (OOO)**: Geeft de basispositie van de servo aan.

Het neurale netwerk dient deze parameters te leren en af te stemmen zodat de gewenste bewegingen worden gegenereerd.

**2. Wiskundig model van een CPG**

De uitgangspositie van een servo wordt gemodelleerd als:

Positie(t)=A⋅sin (2πft+ϕ)+O

Voor 3 servo's:

Servoi(t)=Ai⋅sin (2πfit+ϕi)+Oi voor i=1,2,3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Servo 1 |  | Servo 2 |  | Servo 3 |  |
| Amplitude |  | Amplitude |  | Amplitude |  |
| Frequentie |  | Frequentie |  | Frequentie |  |
| Phase |  | Phase |  | Phase |  |
| Offset |  | Offset |  | Offset |  |

Lengte

Breedte

Hoogte

M1

M2

COG

Target - >

Servo speed moet aangepast kunnen worden.

Sequence delay() tussen de sequences.

Moet uitgebreid worden naar een delay tussen elke sequence.

Clear sequences

Change sequences

Sequences moeten kunnen worden aangepast. En naar een file geschreven kunnen worden.

Sequences inladen vanuit file .

Opgeslagen in een file of copy paste.

Sequences moeten als een plot getoond worden

array[]

tekst naar array[]

wat hebben we nu, en wat kunnen we nu

Bewaar sequences als movement

Play movement

Ontkoppelen

Op eigen kracht

Movements staan op de arduino

Azure

Plan

What needs to be developed

Agile scrum

Github

Yaml

Test

Package

deploy

**CPG met individuele amplitudes en frequenties per servo**

Als we de beweging van de servo’s in een grafiek uitzetten, zien we voor elke servo een golf beweging met verschil in frequentie amplitude en phase.

We kunnen zo de servo aansturen

Phase limits max min

Amplitude limits max min

Frequency limits max min

#include <Servo.h>

// Aantal servo's

#define NUM\_SERVOS 3

// Servo-objecten

Servo servos[NUM\_SERVOS];

// Configuratie van servo's

int servoPins[NUM\_SERVOS] = {8, 9, 10};         // Pinnen waarop de servo's zijn aangesloten

float phaseOffsets[NUM\_SERVOS] = {0, 2.5 \* PI / 3.0, 0}; // Faseverschuivingen

float frequencies[NUM\_SERVOS] = {1.0, 1.0, 2.0}; // Frequentie in Hz per servo

int amplitudes[NUM\_SERVOS] = {30, 50, 20};       // Amplitudes in graden per servo

int midPoints[NUM\_SERVOS] = {90, 90, 90};        // Middenposities per servo

float timeStep = 0.02;                           // Tijdstap in seconden

void setup() {

  Serial.begin(115200); // Voor Serial Plotter (115200 baud)

  for (int i = 0; i < NUM\_SERVOS; i++) {

    servos[i].attach(servoPins[i]); // Servo's initialiseren

  }

}

void loop() {

  static float t = 0; // Tijdvariabele

  // Bereken en stuur de positie voor elke servo

  for (int i = 0; i < NUM\_SERVOS; i++) {

    float pos = midPoints[i] + amplitudes[i] \* sin(2 \* PI \* frequencies[i] \* t + phaseOffsets[i]);

    servos[i].write(pos);

    // Stuur de positie naar de Serial Plotter

    Serial.print("Servo");

    Serial.print(i + 1);

    Serial.print(":");

    Serial.print(pos);

    Serial.print(" ");

  }

  // Einde van de regel voor Serial Plotter

  Serial.println();

  // Tijd bijwerken

  t += timeStep;

  delay(timeStep \* 1000); // Converteer tijdstap naar milliseconden

}

x\_input = [0.1, 0.5, 0.2]

w\_weights = [0.4, 0.3, 0.6]

threshold = 0.5

def step(weighted\_sum):

    if weighted\_sum > threshold:

        return 1

    else:

        return 0

def perceptron():

    weighted\_sum = 0

    for x, w in zip(x\_input, w\_weights):

       weighted\_sum += x\*w

       print(weighted\_sum)

    return step(weighted\_sum)

output = perceptron()

print("output:  " + str(output) )

output

w\_weights

x\_input

0.1

0.04

0.4

0.5

<

0.31

0.19

0.3

0.5

0.6

0.12

0.2

0.1 \* 0.4 = 0,04

threshold

0.1 \* 0.4 + 0.5 \* 0.3 = 0,19

0.1 \* 0.4 + 0.5 \* 0.3 + 0.2 \* 0.6 = 0,31

**Neuraal net werk**

Wat we weten is dat het een harmonische beweging is

Iedere beweging met zijn eigen amplitude frequentie en phase tov de andere beweging.

Een verandering van frequentie betekend een verandering van amplitude.

Een central pattern generator is een vast liggende beweging.

Eigenschap.

Een andere eigenschap niet een nieuwe iets nieuws bestaat niet in ons universum

Alles staat al vast.

We weten het ideale output wat veranderd er als de robot sneller gaat lopen.

Afweging energie verkrijgen ten koste van energie verlies.

Energie verlies < energie winst

Kun je de code aanpassen zodat deze werkt voor een arduino nano met daaraan verbonden een PCA9685 servo controller zodat ik met de slider een servo kan bewegen. De slider staat op een GUI die geprogrammeerd wordt in visual studio met C++/CLI CLR.

Ja nadat ik met de 3 sliders een positie heb ingesteld wil ik deze drie posities bewaren

Met een klik op de save button. Waarna ik opnieuw met de sliders nieuwe posities kan invoeren.

Klik ik weer op save worden deze posities bewaard en toegevoegd aan de eerder bewaarde posities.

Dit kan ik enkele malen herhalen.

Klik ik vervolgens op een play button worden de bewaarde posities na elkaar afgespeeld en verzonden in een loop.

Mocht je willen leren programmeren begin dan niet met C C++ beter om te beginnen met Java

Begin ook eerst met een console en maak zo snel mogelijk je eigen kleine programma’s

Data types

Wat is declareren

Wat betekend initialiseren

Variabelen

Pointers

Stack and heap

Debug

Methodes functions

Parameter argument

Arithmetic

If else

For next

Array[ ]

Struct

library

Object

Van positie naar positie binnen een traject

Max min zelfde weg terug

Richting snelheid

Servo positie ------ > eind positie

Servo positie eind positie

Servo positie eind positie

Kinematisch model vertelt nog niet wat het tip- over punt is

Hiervoor zullen we moeten werken richting een dynamisch model.

Graphic User interface for the PCA9685 servo controller.

Wie kan helpen deze gui te verbeteren?

Bedoeling is dat met een slider de positie van een servo ingesteld kan worden.

Zijn de positie correct wordt de positie met een save button opgeslagen in een txt bestand.

In C++/CLI

Phython

**Product Feature**

* Build Your Own Crawling Robot: This DIY STEM kit lets you assemble a quadruped spider robot from scratch, giving you hands-on experience with robotics and Arduino programming.
* Arduino & ESP8266 Compatibility: The Bionic Quadruped Spider Robot Kit is designed to work with popular Arduino boards and ESP8266 Wi-Fi modules, offering flexibility and expandability for your projects.
* Crawling Action & Wireless Control: Program your robot to move, turn, and navigate using your Arduino code. The ESP8266 module allows for wireless control via Wi-Fi, letting you command your spider robot from afar.
* Educational & Engaging: This robot kit is a fantastic way to learn about robotics, electronics, and programming in a fun and interactive way. Perfect for students, hobbyists, and anyone interested in STEM fields.
* Green & Durable: The robot kit features a green color scheme and is constructed from high-quality materials for long-lasting durability.

This Bionic Quadruped Spider Robot Kit is an exciting DIY project for Arduino enthusiasts and STEM learners. It's a perfect combination of robotics, electronics, and programming, allowing you to build a functional spider robot that crawls and explores its surroundings. The kit includes all the necessary components, including the ESP8266 NodU module for Wi-Fi connectivity, allowing you to control your spider robot remotely. This kit is ideal for anyone interested in learning about robotics, programming, and the Internet of Things (IoT). With its user-friendly design and detailed instructions, you can easily assemble and program your own unique spider robot. Whether you're an experienced maker or a curious beginner, this kit offers a fun and rewarding learning experience. Get ready to unleash your creativity and build a robotic spider that will amaze you and your friends.

Jasmijn gefeliciteerd met je hatchling

De enige robot die de oorsprong van de loopbeweging in zich heeft.

Denk aan toen je een baby was

En je ervan droomde te kunnen lopen.

Net als een blauwdruk of algoritme zul je netjes al die stappen moeten doorlopen, die al beschreven staan in je DNA.

Met de Gui kunnen we de servo posities instellen en als sequence opslaan.

De sequence kunnen we afspelen serial en als text file opslaan.

De text file tekst kunnen we copieren en als progmem pasten in de arduino code

De robot kan dan lopen zonder seriele verbinding met pc.

Via bluetooth of wifi kunnen de sequences in verschillende volgordes afgespeelt worden.

Verzenden van seriele tekst

S1:90,Speed=20

Wat nodig is, is servo nummer positie en snelheid.

11 x 3 waardes byte 1 byte 2 byte 3

Sequence

[byte1][byte2][byte3]

[byte1][byte2][byte3]

[byte1][byte2][byte3]

[byte1][byte2][byte3]

[byte1][byte2][byte3]

[byte1][byte2][byte3]

[byte1][byte2][byte3]

[byte1][byte2][byte3]

[byte1][byte2][byte3]

[byte1][byte2][byte3]

[byte1][byte2][byte3]

End Sequence

01 060 05

02 090 10

03 120 40

060 05 090 10 120 40

Uart verbinding met arduino

Data invoeren in een tekst box

Data verzenden naar arduino

Data opslaan in textbox als serie

Data opslaan als txt bestand

Data ophalen txt bestand

Data aanpassen

Data in grafiek tonen

SEQUENCE 1

|  |  |  |
| --- | --- | --- |
| Positions 1 | | |
| servo | positie | snelheid |
| 01 | 30 | 6 |
| 02 | 120 | 30 |
| 03 | 93 | 12 |
| 04 | 22 | 8 |
| 05 | 45 | 2 |
| 06 | 120 | 9 |
| 07 | 160 | 12 |
| 08 | 120 | 30 |
| 09 | 93 | 12 |
| 10 | 22 | 8 |
| 11 | 45 | 2 |

|  |  |  |
| --- | --- | --- |
| Positions 2 | | |
| servo | positie | snelheid |
| 01 | 30 | 6 |
| 02 | 120 | 30 |
| 03 | 93 | 12 |
| 04 | 22 | 8 |
| 05 | 45 | 2 |
| 06 | 120 | 9 |
| 07 | 160 | 12 |
| 08 | 120 | 30 |
| 09 | 93 | 12 |
| 10 | 22 | 8 |
| 11 | 45 | 2 |

Save positions

Position1 komt op regel 1

Position2 komt op regel 2

Etc.

Positions 1 30 6 120 30 93 12 22 8 45 2 120 9 160 12 120 30 93 12 22 8 45 2 end

Positions 2 30 6 120 30 93 12 22 8 45 2 120 9 160 12 120 30 93 12 22 8 45 2 end

Positions 3 30 6 120 30 93 12 22 8 45 2 120 9 160 12 120 30 93 12 22 8 45 2 end

Positions 4 30 6 120 30 93 12 22 8 45 2 120 9 160 12 120 30 93 12 22 8 45 2 play

play

Send data: sequence

Verstuur serial naar arduino.

Stuur regel 1

Stuur regel2

Arduino ontvang data tot en sla op sd. write

Data speel direct

Data save in array

Data write to sd

Arduino start speel sequence read

Arduino stop stop sequence

Data ontvangen

Play data

Ontvang serial arduino

    if (Serial.available() > 0) {

        String input = Serial.readStringUntil('\n');

        input.trim();

Goed we maken nu een GUI programma in Visual Studio C++/CLI

Waarin we de servo posities en snelheid invoeren

Een save button die de posities opslaat in een textBox

Een send button die de Sequence verstuurt naar de arduino.

Stap 1 uart

private: System::Void btnConnect\_Click(System::Object^ sender, System::EventArgs^ e) {

try {

if (btnConnect->Text == "Connect") {

if (sp->IsOpen) {

sp->Close();

}

sp->PortName = cbPort->Text;

sp->BaudRate = Int32::Parse(cbBaud->Text);

sp->Open();

btnConnect->Text = "Disconnect";

}

else if (btnConnect->Text == "Disconnect") {

sp->Close();

btnConnect->Text = "Connect";

}

}

catch (...) {

MessageBox::Show("Something is error");

}

}

Stap 2 serial data

String^ message = txtMessage->Text;

sp->WriteLine(message);

We beginnen met het versturen van serial data als string naar de arduino.

De arduino ontvangt deze string en kan deze lezen.

Nu zullen we de arduino nog duidelijk moeten maken wat het met deze string moet doen.

Het moet uit de ontvangen string moeten kunnen opmaken

Welke actie het moet uitvoeren.

Om het voor de arduino zo makkelijk mogelijk te houden kunnen we een zo’n duidelijk mogelijk bericht sturen.

Een protocol

Door eerst een commando te sturen

Aan de hand van het commando weet de arduino dan wat deze moet uitvoeren.

We typen een positie in de textbox en de speed

Vervolgens bewaren we dit in een messagetextbox

Als sequence1

De message tekst kunnen we nu serial versturen hier kunnen we aanpassingen in aanbrengen voordat we dit versturen.

Command data

Elf servo posities en elf servo speed totaal 22 karakters.

Een sequence bestaat uit 22 karakters spatie

Wat als we andere tekst sturen de arduino moet dus een start code ontvangen zodat deze weet

Hoe het de karakters moet lezen.

b.v. begint de regel met sequence dan lees de volgende karakters zo

begint het met stop lees de karakters dan zo.

Lees van de sd kaart.

Start code dan lees zoveel bytes als :

Stop code

Stap 3 timer

62 63 64

Kerstkaart

Robert Sjacko

Fabian

Isabelle

Jasmijn

Arthur

falconrui 59 kamer 9

2000 Antwerpen

Gea Jimmy

Bart Mira Julia

Hans Marlies

Anne konijn pleur en nijntje

Soraya Peter Kelvin Quinten Eef

Estrelle

Jennifer

Sorteren

Robots

Tutorials

Website

Video

Lopen wat is lopen

Het is een samenstelling van verschillende samenwerkende bewegingen.

Wat is de bedoeling van lopen.

Voortbeweging

Waarom om aan energie te komen.

Oh wat leuk een kaartje van Jennifer,

Let op! Denk van wie zou dat kaartje nu zijn.

Het eertse waar ik aan dacht, …….. Denk van Jennifer!

En ja hoor, Je ziet ik ben je niet vergeten hoor!

Hoop dat alles goed met je is, hoorde dat je toch een zware periode moest door maken.

en verwerken. Hoop dat je alles een plekje hebt kunnen geven.

Zelf heb ik een zwaar ongeval gehad vijf jaar geleden, maar er gelukkig goed van afgekomen.

Zit nu al lekker vijf jaar thuis (anders gezegd heb al vijf jaar vakantie ) en zo als het er naar uit ziet kan ik het nu uitbreiden tot mijn pensioen.

Ja heerlijk is dat.Nadeel kan geen verlof meer op pakken of vakantie nemen.

Heb wel nog een maandje gewerkt met autistische jongeren.

Maar gaat het verder met Jennifer? Nog leuke mooie trein reizen gemaakt?

Wimbledon.

Of wat lijkt je leuk om nog te doen.

Hoe gaat het op je werk nog problemen met (hardwerkende)collega’s.

Laat je niet gek maken.

De volgende tekst wil ik serial naar arduino sturen.

Positions 1 30 6 120 30 93 12 22 8 45 2 120 9 160 12 120 30 93 12 22 8 45 2 end

Positions 2 30 6 120 30 93 12 22 8 45 2 120 9 160 12 120 30 93 12 22 8 45 2 end

Positions 3 30 6 120 30 93 12 22 8 45 2 120 9 160 12 120 30 93 12 22 8 45 2 end

Positions 4 30 6 120 30 93 12 22 8 45 2 120 9 160 12 120 30 93 12 22 8 45 2 play

Deze moet het in een array opslaan.

Hierbij is regel 1 de eerste posities en speed van de servo’s

Dus positions 1 servo 1 positie 30 speed 6 servo 2 positie 120 speed 30 etc.

Arraya1[0] = position servo 1

Array1[1] = speed servo 1

Array1[2] = position servo 2

Array1[3] = speed servo 2

Array1[4] = position servo 3

Array1[5] = speed servo 3

Array1[6] = position servo 4

Array1[7] = speed servo 4

……………………………………….

Array1[21] = position servo 11

Array1[22] = speed servo 11

De volgende regel positions 2 moeten dan in een volgende array2 opgeslagen worden.

Arraya2[0] = position servo 1

Array2[1] = speed servo 1

Array12[2] = position servo 2

Array2[3] = speed servo 2

Array2[4] = position servo 3

Array2[5] = speed servo 3

Array2[6] = position servo 4

Array2[7] = speed servo 4

……………………………………….

Array2[21] = position servo 11

Array2[22] = speed servo 11

Dus de arduino moet eerst bepalen welke data ontvangen wordt en het dan in de array bewaren.

Zodat als alle data ontvangen is de data afgespeeld kan worden door de servo’s in een loop

door door de arrays te lopen

Kun je me arduino code laten zien.

Serial data ontvangen

Serial data direct afspelen

Serial data opslaan

Serial data afspelen van sd

Serial data opslaan op sd

Kies data

Start afspelen afspelen

Stop afspelen

Als serial data available

lees serial data naar de input buffer

tot \n

Roep de functie procescommand(inputbuffer) op

Trim de string en kijk waar de string mee begint.

MOVE Lees de string uit en voer uit

Roep de functie op

waarmee de string begint.

STORE\_PLAY Lees de string uit en bewaar in arrays voer uit PLAY\_PLAY

STORE\_SD -> create new file store strings

PLAY\_SD -> open file name play strings

STOP -> interrupt beeindig de loop

START -> open file name forward start de begin beweging.

Bluetooth

Forward lees welke beweging bezig is ga via beweging x naar forward.

Backward

Turn left

Turn right

Left

Right

Fast

Slow

autonoom

Obstruction stop move backward turn move forward.

Ontvangen van serial data

Baudrate

Serial available

Buffer

Verstuur ontvangen serial data

Bewaar serial data

Haal de data eruit die we nodig hebben.

Data opgeslagen

Ga naar de functie speel data af

While (true)

For(i = 0 ; i < 11; i++)

Servo 0 positoin[0] speed[0]

If I == 11

I = 0

const char separator = ',';

str.indexOf(separator) geeft de index van de eerste komma (,) in de string.

str.substring(0, index) retourneert het gedeelte van de string vóór de komma.

.toInt() converteert dit naar een integer.

str = str.substring(index + 1);

str begint nu bij index 4

================================================================================

Wat je ziet is dat de servo’s harmonisch oscilleren elk met een eigen amplitude frequentie phase

Verder zien we dat 2 dezelfde frequentie hebben en servo 3 de dubbele frequentie van servo 1

De verhouding Amplitude is, servo1 : servo 2 : servo 3 is 20: 30 : 9

Servo 1 = A sin (2 πft + Phase1)

Servo 2 = B sin (2 πft + phase2)

Servo 3 = C sin (2\*2 πft + phase3)

Het juiste verschil in Phase bepaalt de ideale beweging.

Hoe maak ik hier een cpg van, laat zo gedetailleerd mogelijk zien welke stappen ik moet doorlopen om hier een cpg van te maken. Of wat er nog voor nodig is.

B bepaald door A berekend door lengte breedte hoogte DNA groei veranderd

B bepaald door tip-over punt

#include <Wire.h>

#include <Adafruit\_PWMServoDriver.h>

#include <SD.h>

Adafruit\_PWMServoDriver pwm = Adafruit\_PWMServoDriver();

#define SERVOMIN 150

#define SERVOMAX 600

#define SD\_CS\_PIN 4 // Chip Select pin voor SD-kaartmodule

// Functiedeclaraties

void processCommand(String command);

void executeMove(String command);

void executeStorePlay(String command);

void executeStoreSD(String command);

void executePlaySD(String command);

void executeWriteOver(String command);

void parseServoData(String data);

void playServoData();

bool writeToSD(String fileName, String data, bool overwrite = false);

bool readFromSD(String fileName);

// Variabelen

String inputBuffer = "";

int servoData[11][2]; // Opslag voor servo posities en snelheid

File sdFile;

void setup() {

  Serial.begin(9600);

  Serial.println("Arduino met Command Protocol en SD-kaart!");

  // PCA9685 initialiseren

  pwm.begin();

  pwm.setPWMFreq(60);

  // SD-kaart initialiseren

  if (!SD.begin(SD\_CS\_PIN)) {

    Serial.println("SD-kaart initialisatie mislukt!");

    while (1);

  }

  Serial.println("SD-kaart gevonden!");

}

void loop() {

  // Verwerk serial input

  if (Serial.available()) {

    char receivedChar = Serial.read();

    if (receivedChar == '\n') {

      processCommand(inputBuffer);

      inputBuffer = ""; // Maak buffer leeg

    } else {

      inputBuffer += receivedChar; // Voeg karakter toe aan buffer

    }

  }

}

void processCommand(String command) {

  command.trim();

  if (command.startsWith("MOVE")) {

    executeMove(command);

  } else if (command.startsWith("STORE\_PLAY")) {

    executeStorePlay(command);

  } else if (command.startsWith("STORE\_SD")) {

    executeStoreSD(command);

  } else if (command.startsWith("PLAY\_SD")) {

    executePlaySD(command);

  } else if (command.startsWith("WRITE\_OVER")) {

    executeWriteOver(command);

  } else {

    Serial.println("Onbekend commando!");

  }

}

// Commandoverwerking

void executeMove(String command) {

  int servoNum = command.substring(command.indexOf(',') + 1, command.indexOf(',', command.indexOf(',') + 1)).toInt();

  int position = command.substring(command.indexOf(',', command.indexOf(',') + 1) + 1, command.lastIndexOf(',')).toInt();

  int speed = command.substring(command.lastIndexOf(',') + 1).toInt();

  if (servoNum >= 1 && servoNum <= 11) {

    int pulseWidth = map(position, 0, 180, SERVOMIN, SERVOMAX);

    pwm.setPWM(servoNum - 1, 0, pulseWidth);

    Serial.print("Servo ");

    Serial.print(servoNum);

    Serial.print(" beweegt naar positie ");

    Serial.print(position);

    Serial.print(" met snelheid ");

    Serial.println(speed);

  }

}

void executeStorePlay(String command) {

  int startIdx = command.indexOf(',') + 1;

  String data = command.substring(startIdx);

  parseServoData(data);

  playServoData();

}

void executeStoreSD(String command) {

  int startIdx = command.indexOf(',') + 1;

  String fileName = command.substring(startIdx, command.indexOf(',', startIdx));

  String data = command.substring(command.indexOf(',', startIdx) + 1);

  if (writeToSD(fileName, data)) {

    Serial.println("Data opgeslagen op SD-kaart: " + fileName);

  } else {

    Serial.println("Opslag mislukt!");

  }

}

void executePlaySD(String command) {

  int startIdx = command.indexOf(',') + 1;

  String fileName = command.substring(startIdx);

  if (readFromSD(fileName)) {

    Serial.println("Data gelezen en afgespeeld van SD-kaart: " + fileName);

  } else {

    Serial.println("Kan bestand niet lezen!");

  }

}

void executeWriteOver(String command) {

  int startIdx = command.indexOf(',') + 1;

  String fileName = command.substring(startIdx, command.indexOf(',', startIdx));

  String data = command.substring(command.indexOf(',', startIdx) + 1);

  if (writeToSD(fileName, data, true)) {

    Serial.println("Data overschreven op SD-kaart: " + fileName);

  } else {

    Serial.println("Overschrijven mislukt!");

  }

}

// Hulpfuncties

void parseServoData(String data) {

  int startIndex = 0, dataIndex = 0;

  while (dataIndex < 22 && startIndex < data.length()) {

    int nextComma = data.indexOf(',', startIndex);

    if (nextComma == -1) nextComma = data.length();

    int value = data.substring(startIndex, nextComma).toInt();

    servoData[dataIndex / 2][dataIndex % 2] = value;

    startIndex = nextComma + 1;

    dataIndex++;

  }

}

void playServoData() {

  for (int i = 0; i < 11; i++) {

    int pulseWidth = map(servoData[i][0], 0, 180, SERVOMIN, SERVOMAX);

    pwm.setPWM(i, 0, pulseWidth);

    delay(servoData[i][1] \* 10);

  }

}

bool writeToSD(String fileName, String data, bool overwrite = false) {

  if (overwrite) {

    sdFile = SD.open(fileName.c\_str(), FILE\_WRITE);

  } else {

    sdFile = SD.open(fileName.c\_str(), FILE\_WRITE | O\_APPEND);

  }

  if (sdFile) {

    sdFile.println(data);

    sdFile.close();

    return true;

  }

  return false;

}

bool readFromSD(String fileName) {

  sdFile = SD.open(fileName.c\_str());

  if (sdFile) {

    while (sdFile.available()) {

      String line = sdFile.readStringUntil('\n');

      Serial.println("Gelezen: " + line);

      parseServoData(line);

      playServoData();

    }

    sdFile.close();

    return true;

  }

  return false;

}

**Data naar sd card**

#include <**SD**.h>

#define PIN\_SPI\_CS 4

**File** myFile;

void setup() {

**Serial**.begin(9600);

if (!**SD**.begin(PIN\_SPI\_CS)) {

**Serial**.println(F("SD CARD FAILED, OR NOT PRESENT!"));

while (1); // don't do anything more:

}

**Serial**.println(F("SD CARD INITIALIZED."));

if (!**SD**.exists("arduino.txt")) {

**Serial**.println(F("arduino.txt doesn't exist. Creating arduino.txt file..."));

// create a new file by opening a new file and immediately close it

myFile = **SD**.open("arduino.txt", FILE\_WRITE);

myFile.close();

}

// recheck if file is created or not

if (**SD**.exists("arduino.txt"))

**Serial**.println(F("arduino.txt exists on SD Card."));

else

**Serial**.println(F("arduino.txt doesn't exist on SD Card."));

}

void loop() {

}

SD begin

open (file name, FILE\_WRITE)

open (file name, FILE\_READ)

close

exists

Lopen is een samenwerking van verschillende bewegingen. Die onderling afhankelijk van elkaar zijn.

De vergeten beweging

Doos

Arduino nano 10 stuks = 24,51

PCA9685 10 stuks = 23,65

LM 2596 10 stuks = 1,95

Sd 10 stuks = 10,00

Sg92r 10 x 11 stuks = 250,00

Totaal 10 robots = 310 euro

3d printed onderdelen 30 euro

Schroeven en moeren

draad

Doos

Papierwerk

Totaal 400 euro 10 stuks x 250 2500 euro

Totaal 4000 euro 100 stuks x 250 25000 euro

Totaal 40000 euro 1000 stuks x 250 250000 euro

Boek 100 euro

Ander perspectief

Systimatisch geordend

Algoritme

Sequentieel

Getallen lijn

Vergelijkingen

1

½

-1

Wortels

Log

Exp

Variabelen x y a

1 + 1 = 2

3 + 5 = 8

x + x = 2 \* x

x + y = y + x

**truc vul het in !**

truc schrijf de formule op!

kruislings vermenigvuldigen

tussen haakjes zetten

**regels !**

**ARDUINO**

De arduino zal serial data ontvangen, die de arduino vertelt wat deze moet doen.

Hiervoor zullen we een programma op de arduino moeten hebben.

Nadeel is dat we geen data kunnen opslaan, hiervoor zullen we een sd card aansluiten op de arduino.

Zodat de arduino stand-alone kan functioneren.

We zullen nu stap voor stap een programma schrijven waarmee we kunnen communiceren met de arduino.

De communicatie zal zowel via de pc serial als wel via bluetooth mogelijk zijn. Via bluetooth kunnen we een remote controller gebruiken of een android phone.

**ontvangen van serial data**

nu we serial data kunnen ontvangen, wat we wilen is een commando sturen naar de arduino zodat deze weet wat het moet doen. Om het voor de arduino zo makkelijk mogelijk te maken om de data te lezen maken we een soort protocol waarin we beschrijven hoe en welke commando’s we moeten sturen zodat de arduino weet hoe het dit commando kan uitvoeren.

Zo kunnen we de gestuurde serial data al uit filteren is het

De String class helpt ons hiermee.

Start de serial data met een commando voer dan dit commando uit.

We kunnen nu een lijst maken met verschillende commando’s zoals:

POSITION

PLAY\_SEQUENCE

SAVE\_TO\_SD

PLAY\_FROM\_SD

GET\_FROM\_SD

START

STOP

LEFT

RIGHT

FORWARD

**analyseren van ontvangen data.**

Commando execute functionX

Het eerstse deel van de serial data moet bestaan uit het commando

Laten we nu bekijken wat serial data kunnen manipuleren. In een bepaald format

charAt()

compareTo()

concat()

c\_str()

endsWith()

equals()

equalsIgnoreCase()

getBytes()

indexOf()

lastIndexOf()

length()

remove()

replace()

reserve()

setCharAt()

startsWith()

substring()

toCharArray()

toDouble()

toInt()

toFloat()

toLowerCase()

toUpperCase()

trim()

Deze functies kunnen we gebruiken uit de String class

We maken hier een object van String command =””;

Oproepen functies:

Bluetooth:

Commands

Android

Als serial data available

lees serial data naar de input buffer

tot \n

Roep de functie procescommand(inputbuffer) op

Trim de string en kijk waar de string mee begint.

MOVE Lees de string uit en voer uit

Roep de functie op

waarmee de string begint.

STORE\_PLAY Lees de string uit en bewaar in arrays voer uit PLAY\_PLAY

STORE\_SD -> create new file store strings

PLAY\_SD -> open file name play strings

STOP -> interrupt beeindig de loop

START -> open file name forward start de begin beweging.

=========================== commando POSITION ====================================

Laten we beginnen met POSITION

Servo 1, position, speed

Het commando ziet er dan zo uit:

POSITION, Servo 1, position, speed.

POSITION, 1, 120, 5

Length

We ontvangen de serial data en gaan na of deze begint met POSITION

startsWITH(POSITION) then

function POSITION\_SERVO()

Void Position\_Servo(String str){

Hier ontvangen we de string en ontleden deze.

}

#include <Wire.h>

#include <Adafruit\_PWMServoDriver.h>

#include <SD.h>

Adafruit\_PWMServoDriver pwm = Adafruit\_PWMServoDriver();

#define SERVOMIN 150

#define SERVOMAX 600

// Functiedeclaraties

void processCommand(String command);

void executeMove(String command);

// Variabelen

String inputBuffer = "";

void setup() {

  Serial.begin(9600);

  Serial.println("Arduino met Command Protocol en SD-kaart!");

  // PCA9685 initialiseren

  pwm.begin();

  pwm.setPWMFreq(60);

}

void loop() {

  // Verwerk serial input

  if (Serial.available()) {

    char receivedChar = Serial.read();

    if (receivedChar == '\n') {

      processCommand(inputBuffer);

      inputBuffer = ""; // Maak buffer leeg

    } else {

      inputBuffer += receivedChar; // Voeg karakter toe aan buffer

    }

  }

}

void processCommand(String command) {

  command.trim();

  if (command.startsWith("MOVE")) {

    executeMove(command);

  } else {

    Serial.println("Onbekend commando!");

  }

}

// Commandoverwerking

void executeMove(String command) {

  int servoNum = command.substring(command.indexOf(',') + 1, command.indexOf(',', command.indexOf(',') + 1)).toInt();

  int position = command.substring(command.indexOf(',', command.indexOf(',') + 1) + 1, command.lastIndexOf(',')).toInt();

  int speed = command.substring(command.lastIndexOf(',') + 1).toInt();

  if (servoNum >= 1 && servoNum <= 11) {

    int pulseWidth = map(position, 0, 180, SERVOMIN, SERVOMAX);

    pwm.setPWM(servoNum - 1, 0, pulseWidth);

    Serial.print("Servo ");

    Serial.print(servoNum);

    Serial.print(" beweegt naar positie ");

    Serial.print(position);

    Serial.print(" met snelheid ");

    Serial.println(speed);

  }

}

Test

Error

const char separator = ',';

str.indexOf(separator) geeft de index van de eerste komma (,) in de string.

str.substring(0, index) retourneert het gedeelte van de string vóór de komma.

.toInt() converteert dit naar een integer.

str = str.substring(index + 1);

str begint nu bij index 4

**Sequentie**

We voeren als sequence enkele posities in, die opgeslagen moeten worden

sequence invoeren.

SEQUENCE, 1, 160, 5. Dit is de eerste sequence die opgeslagen moet worden, als

Sequence1, servo 1, positie 90, speed 5.

SEQUENCE, 1, 10, 5. Dit is de tweede sequence die opgeslagen moet worden als

Sequence2, servo 1, positie 90, speed 5. Etc.

Invoer PLAY\_SEQUENCE

De sequence moet nu in een loop afgespeelt worden.

Commando: SEQUENCE :

Get servo nr position speed

Save in array hoeveel sequences willen we bewaren.

Commando: PLAY\_SEQUENCE :

Get array start bij index 0 positie index 2 speed

index 3 positie index 4 speed

belangrijk voordat we de nieuwe positie in geven zal de oude positie bereikt moeten zijn.

We hoeven niet de servo door te geven als deze elkaar opvolgen

Commando: stop\_sequence

Commando: new\_sequence

**SD\_CARD**

Er is onvoldoende geheugen op de arduino om een sequence op te slaan.

Hiervoor gebruiken we een sd card, waarop we dan onze sequences kunnen opslaan.

**BLUETOOTH**

Nu we sequences kunnen opslaan en de arduino stand-alone kunnen laten werken.

Kunnen we commando’s versturen via bluetooth

**SINUS FUNCTION**

Wat we zien is dat elke servo zijn eigen alternating harmonische beweging maakt tesamen vormt dit een movement. Voor de core walk kunnen we berekenen welke beweging elke servo afzonderlijk moet maken met respect voor de snelheid.zie berekening voor de positie van de servo in tijd

Deze kunnen we beschrijven als een sinus functie.

We zullen de servo nou aansturen met behulp van een sinus functie

S1\_Position = A\*SIN(2phift + phase) + offset

S2\_Position = A\*SIN(2phift + phase) + offset

S3\_Position = A\*SIN(2phift + phase) + offset

Amplitude

Frequentie

Phase

Offset

Ja leuk he weer een kaartje

Vond de souvenir die je voor me mee bracht echt super lief,

Het vertelt iets over je kwaliteiten van je.

Je waardevol je bent.

Je vindt jezelf toch wel wat meer waard

Je bent toch wel een heel speciaal meisje.

Laat je zien dat je meer waardt bent

Dat jij niet zo als Jasmijn jongens gaat fixen.

En hoe rijk je al bent. Stijl en klasse hebt.

Rijkdom zit niet alleen in geld.

En als het alleen geld is wat jou geluk kan brengen dan doe je iets fout!

Zorg ervoor dat je ook dit jaar weer leuke reizen kunt maken.

Niet te veel want je hebt nog een heel leven te gaan.

Bewaar dan ook wat voor later.

Het leven neemt en het leven geeft

Jasmijn

Wat ben je aan het doen.

Zal deze week je robot afmaken, moet haar alleen nog wat kunstjes leren.

Zoals boys fixen. Dat valt niet mee vooral omdat ik niet weet hoe dat moet.

Anders leer jij het de robot zelf maar.

struct ServoSequence {

    int positions[MAX\_POSITIONS]; // Reeks van posities

    int speeds[MAX\_POSITIONS];    // Reeks van snelheden

    int numPositions;             // Aantal posities in de reeks

    int currentPosIndex;          // Huidige doelpositie index

    int currentPos;               // Huidige positie

    bool atTarget;                // Status of de servo zijn eindpositie heeft bereikt

    unsigned long lastUpdate;     // Tijdstip van laatste update

};

ServoSequence servos[NUM\_SERVOS] = {

    {{10, 160, 10}, {10, 10, 10}, 3, 0, 20, false, 0},  // Servo 0: Posities en snelheden

    {{10, 160, 60}, {15, 15, 20}, 3, 0, 10, false, 0}, // Servo 1: Posities en snelheden

    {{45, 90, 90}, {5, 5, 5}, 3, 0, 45, false, 0}  // Servo 2: Posities en snelheden

};

Sequence1 = servo1 position speed, servo2 position speed, servo3 position speed

Sequence2 = servo2 position speed, servo3 position speed

Sequence3 = servo1 position speed, servo3 position speed

S1 0 90 5, 1 60 10, 2 120 15 S2 0 90 5, 2 120, 15 S3 1 60 10 2 120 15

Pos[1] = 90 , pos[2] = 60

Speed[1] = 5 , speed[2] = 12

Servo1= pos[1] speed[1]

Servo2= pos[2] speed[2]

Servo3= pos[3] speed[3]

Sequence1 = servo1, servo2, servo3

Sequence2 = servo2, servo3

We beginnen opnieuw let op! servo’s hoeven niet aan gestuurd te worden

Het gaat alleen om het opslaan van de sequences!

Begin met in de monitor de boodschap

PCA ready geef sequences

via de monitor wordt dit serial ingevoerd:

S1 0 90 5 1 60 10 2 120 15 S2 0 120 5 2 10 6 S3 1 60 10 2 120 15

Wat staat voor:

Sequence1 servo1 position speed servo2 position speed servo3 position speed

Sequence2 servo1 position speed servo2 position speed

Sequence3 servo2 position speed servo3 position speed

Deze sequences moeten nu opgeslagen worden op de arduino nano

Zodra deze opgeslagen zijn

Toont de monitor

Sequences opgeslagen

Sequence 1 servo1 position 90 speed 5 servo2 position 60 speed 10 servo3 position 120 speed 15

Sequence 2 servo1 position 120 speed 5 servo3 position 10 speed 6

Sequence 3 servo2 position 60 speed 10 servo3 position 120 speed 15

Verder dus niets doen dan alleen deze code, dus niet de servo’s aansturen!

Ja nu kunnen we verder

Nadat de sequences zijn opgeslagen

Voeren we serial PLAY in.

Sequence1 wordt dan afgespeeld tot dat alle servo’s hun eindposities bereikt hebben.

De servo’s bewegen dan naar hun posities met de snelheid.

Daarna de volgende sequence.

Na de laatste sequence wordt weer begonnen met sequence1.

Voeren we serial STOP in dan stopt het afspelen.

Pas de code die we hebben hiervoor aan

#include <Arduino.h>

#include <Wire.h>

#include <Adafruit\_PWMServoDriver.h>

#define SERVOMIN 150 // Minimale pulsbreedte in ticks

#define SERVOMAX 600 // Maximale pulsbreedte in ticks

#define MAX\_SEQUENCES 10 // Maximaal aantal sequences

#define MAX\_COMMANDS 10 // Maximaal aantal commando's per sequence