

# **Prosjektoppgave i Datateknikk**

## **WALL\_E**

Ryddende beltebilrobot med  
radiokommunikasjon og nettside

**IELET1002**  
**Gruppe 22**  
**Trondheim, vår 2022**

**KANDIDATER (etternavn, fornavn):**

Nordmo, Håkon

Myhre, Gunnar

Virani, André Joseph

DATO:	FAGKODE:	GRUPPE (navn/nr):	SIDER/BILAG:
6. mai, 2022	IELET1002	22	25 / X

**FAGLÆRER:**

Arne Midjo

**TITTEL:**

Prosjektoppgave i Datateknikk

**SAMMENDRAG:**

I dette prosjektet har vi programmert en ZUMO32u4 belterobot ved hjelp av Arduion-IDE. Zumoen har fått flere funksjoner for å kunne bevege seg innenfor et område og hente tomme brusbokser å kjøre de ut av banen. Et av disse programmene er en linjefølger, slik at den kan følge stripene med teip for å finne frem. I tillegg så har den en funksjon for å simulere et batteri (software-batteri), som tar hensyn til hvor langt zumo-bilen har beveget seg. Bilen går også til en bestemt plass for å lade software-batteriet og legge fra seg bokser, en tenkt ladestasjon/miljøstasjon. Oppå zumoen er det montert en ESP32 som kommuniserer med zumo-bilen over serial. Denne ESP-en fungerer som radiosender for Zumoen. Zumoen er utrusta med ultrasonisk sensor for å detektere bokser. En Raspberry Pi kjører Node-RED og kommuniserer med en annen ESP32 over USB. De to ESP-ene i systemet kommuniserer med hverandre ved bruk av ESP now.

# Innhold

<b>1 Innledning</b>	<b>4</b>
<b>2 Terminologier</b>	<b>5</b>
<b>3 Teori</b>	<b>6</b>
3.1 Programmering i Arduino-C . . . . .	6
3.1.1 Arduino IDE . . . . .	6
3.2 Sensorer og tilleggskomponenter . . . . .	7
3.2.1 Ultrasonisk sensor . . . . .	7
3.2.2 Linjesensorer . . . . .	8
3.2.3 Kvadraturenkoder . . . . .	8
3.2.4 Servomotor . . . . .	8
3.2.5 TXS0108E 8-kanals logisk nivåoversetter . . . . .	8
<b>4 Metode</b>	<b>9</b>
4.1 Versjonskontroll for kildekoden . . . . .	9
4.2 Zumo32u4 . . . . .	9
4.2.1 Hardware . . . . .	9
4.2.2 Programstruktur . . . . .	10
4.3 Pakkestruktur for kommunikasjonsprotokoll . . . . .	11
4.4 Seriekommunikasjon . . . . .	13
4.5 ESP32 . . . . .	13
4.5.1 Esp Now . . . . .	14
4.5.2 TTGO OLED-fargeskjerm . . . . .	15
4.6 Ekspansjonskretskort for Zumo . . . . .	15
4.7 Node-Red . . . . .	15
<b>5 Resultat</b>	<b>18</b>
5.1 Generell virkemåte . . . . .	18
5.2 Resultat av test . . . . .	18
<b>6 Diskusjon</b>	<b>20</b>
6.1 Teknologivalg . . . . .	20
6.1.1 Bruk av ESP Now i stedet for MQTT . . . . .	20
6.2 Problemer vi har lært av . . . . .	20
6.2.1 Servomotor og pinouts fra ZUMO32u4 . . . . .	20
6.2.2 Wall-E gikk sporadisk i kalibreringsmodus . . . . .	21
6.3 Forbedringer . . . . .	21
6.3.1 Optimalisering av kommunikasjonsprotokollen . . . . .	21
6.3.2 Direktestyring av zumoen med spillkontroller . . . . .	22
6.3.3 Låsing av last under transport . . . . .	22
6.3.4 Flere ESP32 noder . . . . .	22
6.4 Relevans med tanke på FNs bærekraftsmål . . . . .	22
<b>7 Konklusjon</b>	<b>23</b>

## 1 Innledning

I dette Smart City prosjektet har roboten WALL.E fått i oppdrag å rydde opp byen E-Ville, slik at menneskene en dag kan flytte tilbake. WALL.E tjener penger ved å pante søppel den finner, deretter bruker den pengene på å lade opp batteriene sine. Dette er altså en viktig oppgave som er avgjørende for at menneskene skal kunne reise tilbake til jorden.

WALL.E er bygget på ATmega32U4 mikrokontrolleren, og tar også i bruk en ESP32 mikrokontroller for kommunikasjon med menneskenes romstasjon. Romstasjonen drives av en Raspberry Pi, og kommuniserer med WALL.E ved hjelp av en ESP32.

Infrastrukturen på bakken baserer seg på en motorvei rundt byen, av typen "EL-teip". Roboten kjører rundt i byen og leter etter søppel. Når søppel blir funnet, kjører roboten ut til motorveien og deretter ut av byen til en miljøstasjon der roboten kan pante søppelet sitt, og lade batteriene.

**Vi har også laget en videopresentasjon på 14 minutter som demonstrerer WALL.E [1]**

(Som en liten parantes må vi nevne at grunnen til at André ikke er med på videopresentasjonen er at det skjedde noe uventet hjemme som gjorde at han måtte dra ut av byen.)

## 2 Terminologier

PMW	Pulse width modulation
MCU	Mikrokontroller
ESP32	32-bits MCU med 2,4GHz trådløs kommunikasjon
SG90	PWM-styrt servomotor
ESP Now	Protokoll for radiokommunikasjon med ESP32
HC-SR04	Ultrasonisk avstandssensor
TXS0108E	8-kanals logisk nivåskifter
TTGO	Utviklingskort som inneholder ESP32 og OLED-fargeskjerm
GPIO	<i>General Purpose Input/Output</i>
RX	GPIO-pin for å motta ( <i>receive</i> ) seriekommunikasjonsdata
TX	GPIO-pin for å sende ( <i>transmit</i> ) seriekommunikasjonsdata
<i>zumo</i>	Zumo32U4 er en ferdibygget beltebilrobot laget av Polulu.
RPi	Raspberry Pi 3b+ - <i>ettkorts</i> -datamaskin

## 3 Teori

### 3.1 Programmering i Arduino-C

Arduino[2] er et *åpen-kildekode* maskinvare- og programvareprosjekt som er lisensiert under frie og åpne lisenser. Arduino gjør det lettere å programmere mikrokontrollere ved å strømlinjeforme og automatisere kompilasjons- og overføringsprosessene. Arduino tilbyr et rikt standardbibliotek med objektorienterte abstraksjonslag for vanlige maskinvarekomponenter slik som sensorer og aktuatorer. Arduino bruker en C++-kompilator men med ekstensiv preprosessering som abstraherer bort C++ sin vanlige *main*-funksjon til fordel for to arduinospesifikke funksjoner (*setup* og *loop*), samt linker til Arduinos standardbibliotek.

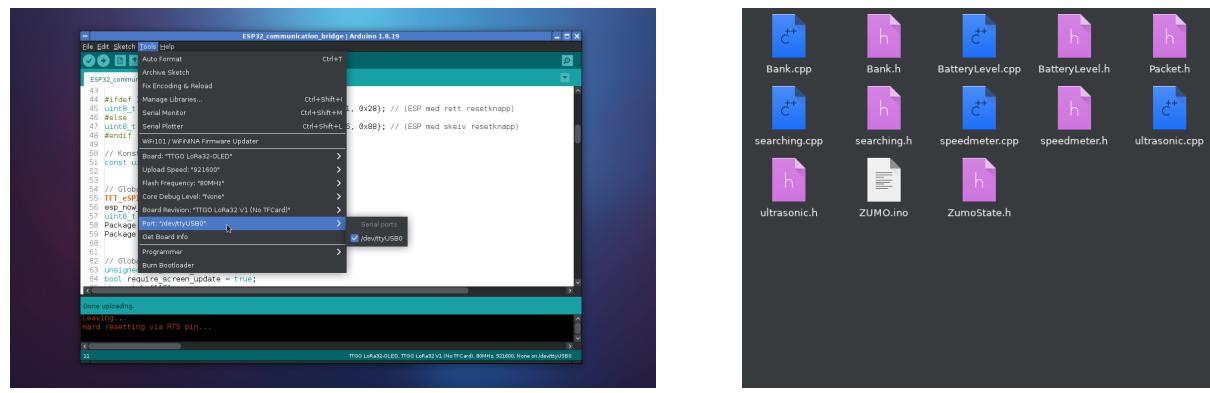


Figure 1: Utvikling ved hjelp av Arduino IDE

#### 3.1.1 Arduino IDE

Arduino IDE (*integrated development environment*) er et integrert utviklermiljø som blant annet tilbyr et skriveprogram, pakkenedlaster for drivere og biblioteker, seriemonitorterminal, integrert opplastingsfunksjon, automatisk kodeformatering og annet.

## 3.2 Sensorer og tilleggskomponenter

### 3.2.1 Ultrasonisk sensor

En ultrasonisk sensor brukes for å måle distanser ved hjelp av lydbølger. Sensoren produserer en lydfrekvens utenfor det menneskelige hørselsområdet, og måler tiden det tar før lydfrekvensen treffer sensoren igjen. Når en vet tiden det tar å få ekkoet tilbake, og en vet hastigheten på lydbølger, kan distansen regnes ut med følgene formel.

$$d = v * t$$

I dette prosjektet brukes HCSR04[3] fra Cytron Technologies. Denne ultrasoniske sensoren er både en transmitter og mottaker i samme komponent. Sensoren kobles enkelt opp til en mikrokontroller med GND, VCC, TRIG, og ECHO pinner. TRIG pinnen kobles til en av mikrokontrollerens GPIO pinner som en OUTPUT, og ECHO pinnen kobles en GPIO pinne som en INPUT.

Power Supply:	+5V DC
Quiescent Current:	<2mA
Working current:	15mA
Effectual Angle:	<15°
Ranging Distance:	2-400 cm
Resolution:	0.3 cm
Measuring Angle:	30°
Trigger Input Pulse width:	10uS
Dimension:	45mm x 20mm x 15mm
Weight:	approx. 10 g



(a) HCSR04

Figure 2: Spesifikasjonene ovenfor er hentet fra brukermanualen til HCSR04

For å sette i gang en distansemåling med HCSR04, sendes en HØY puls med varighet i  $10\mu s$  fra mikrokontrolleren. Når sensoren mottar pulsen på TRIG pinnen vil sensoren produsere 8 raske perioder av en frekvens på 40 kHz. Lydbølgene treffer nærmeste objekt fra sensoren og ekkoet slår tilbake til sensoren. Når sensoren mottar ekkoet, vil den sette ECHO pinnen HØY i like lang tid som lydbølge reiste frem og tilbake. I likningen under vises hvordan en kan beregne distansen i *cm*.

$$distanse = \frac{343 \times 100 \times 10^{-6} \times t}{2} = \frac{0.0343 \times t}{2} \quad (1)$$

### 3.2.2 Linjesensorer

I prosjektet tas det i bruk et brett med fem linjesensorer (GP2S60) festet på undersiden av en ZUMO32u4[4]. Disse linjesensorene bruker infrarødt lys for å kunne skille mellom mørke og lyse overflater, og kan for eksempel brukes i linjefølgingsprogrammer. Hver linjesensor består av en IR-LED og en fotoresistor. Der IR-LED sender ut infrarødt lys, og fotoresistoren måler hvor mye lys som reflekteres tilbake.

### 3.2.3 Kvadraturenkoder

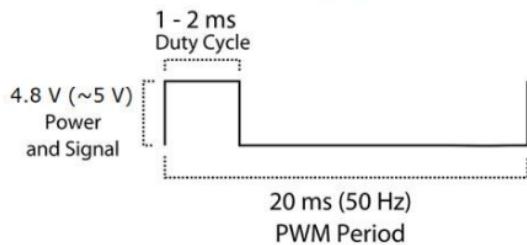
ZUMO32u4 kommer ferdig montert med et sett "Quadrature encoders", disse brukes til å telle rotasjoner på motoraksen. I enden av hver motoraksling er det festet en magnetisk disk, som roterer i forholdet 75:1 til DC-motoren. Den magnetiske disken har en oppløsning på 12 counts per resolution.

$$75.81 \times 12 = 909.7CPR \quad (2)$$

### 3.2.4 Servomotor

**SERVO MOTOR SG90** En servo motor gjør det mulig å styre nøyaktige rotasjonshastigheter og vinkler. Servomotoren som tas i bruk i prosjektet er av typen SG90, og kan roteres  $180^\circ$ .

Servomotoren styres enkelt med én GPIO pinne som kan sende PWM-signaler. For å bestemme hvor mange grader servomotoren skal roteres sender man et PWM-signal på 50 Hz, som tilsvarer en periode på 20ms. I løpet av den perioden skal mellom 1-2ms være HØY, der 1ms tilsvarer  $90^\circ$  mot klokka, og 2ms tilsvarer  $90^\circ$  med klokka.



### 3.2.5 TXS0108E 8-kanals logisk nivåoversetter

**TXS0108E**[6] er en åtte-kanals bidireksjonell logisk nivåoversetter som oversetter aktivt mellom to logiske spenningsnivåer. Referansespenninger kobles opp til de to referansespenningsportene  $V_{CCA}$  (1,4V – 3,6V) og  $V_{CCB}$  (1,65V – 5,5V), dette fordrer at spenningsnivået  $V_{CCA} \leq V_{CCB}$ . GND er det felles jordpunktet til de to referansespenningene. OE (*output enable*) kobles til høyt logisk signal  $V_{CCA}$  for å aktivere nivåoversetteren, settes OE til jord går nivåoversetter i en høyimpedant modus.

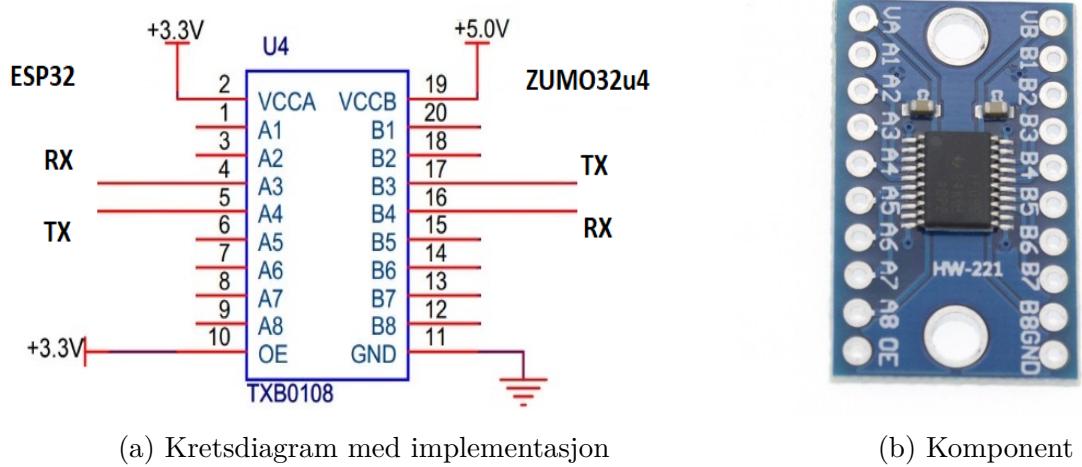


Figure 3: TXS0108E

## 4 Metode

### 4.1 Versjonskontroll for kildekoden

Gruppen startet tidlig å skrive all kode inn i GIT, for å sikre seg et strukturert program fra start. GITmappa ble delt inn i tre deler. En mappe til hver del av prosjektet. Vi brukte GitHub til som vert for GIT-prosjektet [11].

### 4.2 Zumo32u4

Beltebilroboten Zumo32u4 har vært sentral i prosjektet. Den bruker den Arduino-kompatible mikrokontrolleren ATmega32U4 fra Atmel, og bruker et ferdigskrevet bibliotek (*Zumo32U4.h*) tilpasset roboten[5]. Biblioteket inneholder en rekke funksjoner som gjør det enklere ta i bruk hardwaret på Zumo32u4.

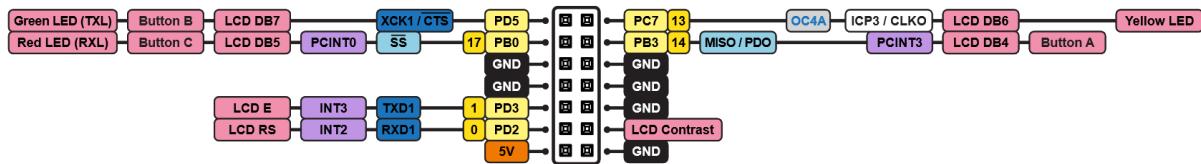
#### 4.2.1 Hardware

WALL.E er satt sammen av komponentene i listen i figur 4. Alle komponentene er montert på Zumo32u4 ved hjelp at et utviklerkort og en sponplate. Utviklerkortet er koblet opp til de seks tilgjengelige pinnene på Zumoen, og kobles videre til ESP32, HCSR04 og Micro Servo 9g.

ESP32 får spenning fra "5V" porten på Zumoen og går gjennom en levelshifter, slik at ESPen mottar en 3.3V spenning. TX(1) og RX(0) pinnene på Zumoen er også koblet via levelshifteren og videre til pinnene TX(17) og RX(2) på ESPen. TX og RX pinnene

brukes til å sende og motta seriekommunikasjonsdata mellom Zumo32u4 og ESP32.

Den ultrasoniske avstandssensoren HCSR04, samt seriellkommunikasjonspinnene RX og TX kobles opp til



nr.	komponentnavn	beskrivelse
0	Pololu Zumo 32U4	mikrokontrollerrobot
1	LILYGO® TTGO T-Display ESP32	esp32 mikrokontroller med skjerm
2	Micro Servo 9g	servomotor 180 grader
3	HCSR04	Ultrasonisk avstandsmåler
4	Utviklerkort	serial kobling mellom ESP og zumo
5	CJMCU-18 (TXS0108E-klone)	Levelshifter: omgjør spenning

Figure 4: Oversikt over komponentene på WALL.E

#### 4.2.2 Programstruktur

På zumoen har vi strukturert hovedprosedyren som en tilstandsmaskin (se figur 5 og 6 for hhv. tilstandsliste og flytskjema). Zumoen vil være i kun én forhåndsdefinert tilstand om gangen, og siden denne rapporteres over til *Node-Red* har vi ganske mye informasjon om hva som foregår på zumoen til enhver tid.

Tilstandsmaskinen er implementert i Arduino-C med en *switch-case* direkte i programmets hovedløkke, som befinner seg i **ZUMO.ino**. Inne i de forskjellige tilstandene har vi unngått å bruke løkker og *delay*-kall slik at programmet ikke skal stoppe eksekvering. Før tilstandsmaskinen oppdateres gjør vi en del oppgaver slik som å motta pakke over serieporten, og gjøre regelmessige sensormålinger. Etter tilstandsmaskinen oppdateres ser vi om det er nødvendig å gjøre en pakkeoverføring til *Node-Red*, dette bestemmes av et flagg (*require\_package\_transmission*) som blir avgjort for hver iterasjon i hovedløkka. Denne struktureringen lar oss også strømlinjeforme en del logiske operasjoner slik som å starte en tidsaker (*time\_in\_state*) hver gang vi endrer tilstand. Vi kan også kjøre operasjoner kun første gang zumoen inntrer i en ny tilstand vha. flagget *state\_has\_changed*, som blir avgjort for hver iterasjon i hovedløkka.

Noen av funksjonalitetene har blitt delt opp i ulike headerfiler som så inkluderes i hovedprogrammet. Dette har spesielt vært hjelpsomt for funksjonalitet som deles mellom Zumo

nr.	tilstandsnavn	beskrivelse
0	RESET	starttilstand, tilbakestiller variabler
1	CALIBRATE_LINESENSORS	kalibreringsprosedyre
2	WAIT_FOR_START_SIGNAL	venter på startsignal
3	SEARCHING_FOR_BOX	kjører rundt og ser etter bokser ("ping-pong")
4	BACKING_FROM_BORDER	rygger vekk fra kanten av innhegninga
5	SMALL_TURN	snur for å unngå å kjøre inn i veggen igjen
6	SCANNING_FOR_BOX	snur rundt og ser etter bokser ("radar")
7	FOUND_BOX	første tegn til boks
8	TURNING_TO_BOX	snur seg direkte mot boks
9	LOST_TRACK_OF_BOX	ser ikke lenger boks
10	MOVING_TO_BOX	beveger seg sakte mot boks
11	GRABBING_BOX	(ubrukt)
12	MOVE_TO_BORDER	tar med seg boks til linja
13	RETURN_TO_STATION	følger linja tilbake til ladestasjon
14	STOPPED	står stille (tom for batteri)
15	REFUELING	lader opp
16	FOLLOW_LINE	kun linjefølging (for utvikling)
17	RETURN_TO_CITY	returnerer tilbake til innhegning

Figure 5: Oversikt over tilstandene, som definert i **ZumoState.h**.

og ESP, slik som **ZumoState.h** og **Packet.h**, for å hindre duplisering av kode – noe som ville gjort vedlikehold vanskelig siden endringer måtte blitt gjort flere plasser.

### 4.3 Pakkestruktur for kommunikasjonsprotokoll

Kommunikasjonen hele veien fra zumoen til Raspberry Pi bruker det samme pakkeformatet på 21 byte (se figur 7). Denne pakka er konstruert i Arduino-C med en *struct*, som lar oss sette sammen et vilkårlig antall variabler av forskjellige datatyper til én sammenhengende datastruktur. Denne pakkesammensetningen er definert i **Packet.h** og brukes i filene **ZUMO.ino** og **ESP32\_communication\_bridge.ino**. I Node-Red er denne pakkestrukturen gjengitt og tolket av en *buffer-parser-node*.

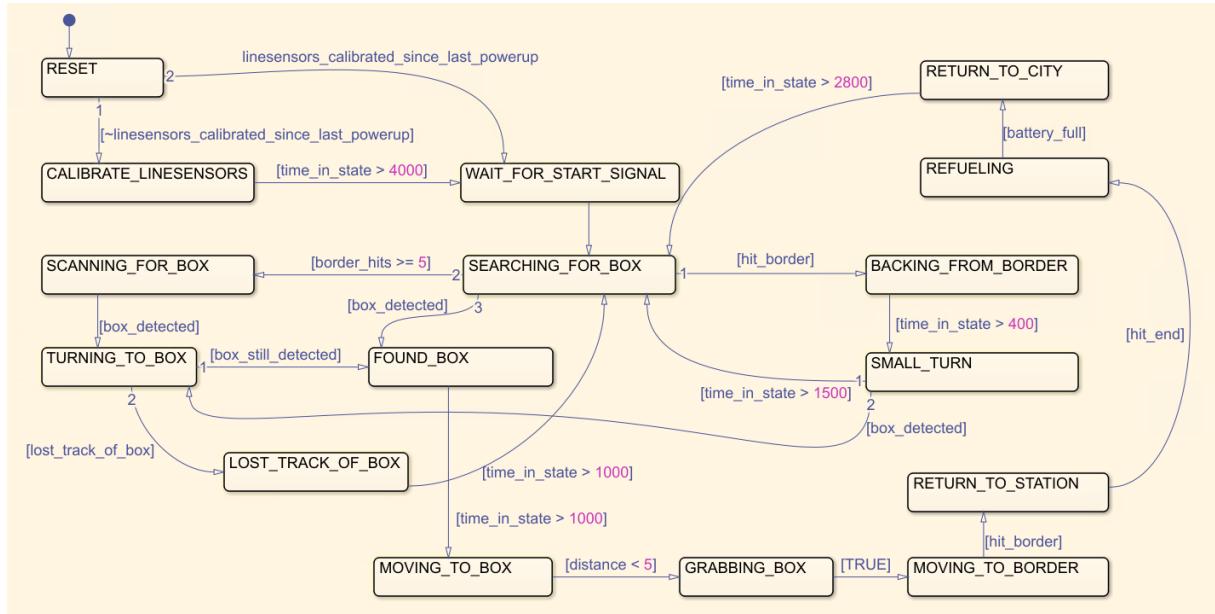


Figure 6: Flytdiagram for tilstandsmaskinen på Zumoen

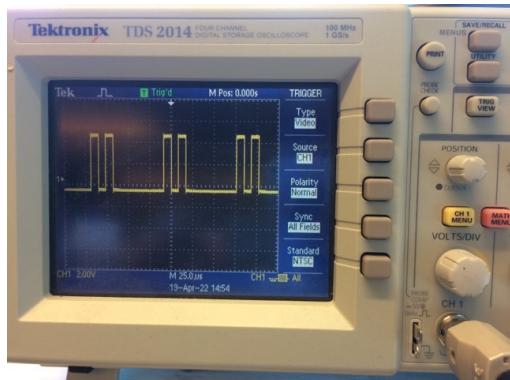
byte	navn	datatype	lengde [byte]	beskrivelse
0	start_byte	byte	1	"<"
1	zumo_state	uint8_t	1	Zumoens tilstand
2	Kp	uint8_t	1	PID-konstant P
3	Ki	uint8_t	1	PID-konstant I
4	Kd	uint8_t	1	PID-konstant D
5	battery_level	uint16_t	2	data
7	speed	uint16_t	2	data
9	ultrasonic_distance_reading	float	4	data
13	update_zumo_state	bool	1	oppdateringsflagg
14	update_Kp	bool	1	oppdateringsflagg
15	update_Ki	bool	1	oppdateringsflagg
16	update_Kd	bool	1	oppdateringsflagg
17	battery_real	uint16_t	2	data
19	bank	uint16_t	2	data
20	stop_byte	byte	1	">"

Figure 7: Oversikt over kommunikasjonspakka, som definert i **Packet.h**.

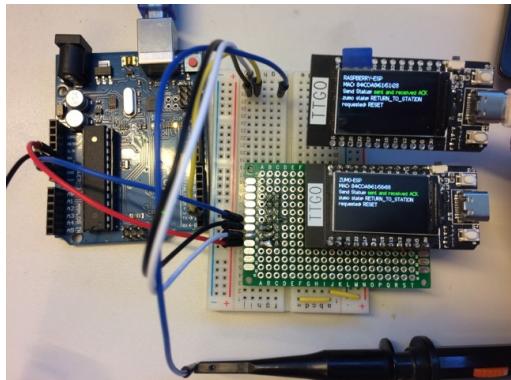
## 4.4 Seriekommunikasjon

Kommunikasjonen mellom **Zumo**↔**Zumo-Esp** og **RPi-Esp**↔**RPi** blir overført over én sendepinne (TX) og én mottakerpinne (RX) som er krysskoplet mellom to og to apparater. På Zumoen og ESP bruker vi det forhåndsdefinerte Arduinobiblioteket til å sende og motta data, mens på Raspberry Pi bruker vi Node-Red sin *serial-node*. Vi bruker en baudrate på 115200[bit/s], og det logiske nivået er 3,3V – med unntak av Zumoen som bruker 5V. Denne nivåforskjellen løste vi med en logisk nivåskifter. Under utviklingen testet vi med oscilloskop for å bekrefte at de logiske nivåene ble korrekt oversatt (se figur 8). Spesielt viktig var det at ESP-en ikke fikk inn 5V, siden det ville kunne skadet chippen.

Apparat	TX	RX	logisk nivå	interface	baudrate
Zumo32u4	0(PD2)	1(PD3)	5V	Serial1 (Arduino)	115200
Zumo-Esp	17	2	3,3V	Serial (egedefinert)	115200
RPi-Esp	USB	USB	3,3V	Serial (Arduino)	115200
Raspberry Pi	USB	USB	3,3V	/dev/ttyUSB0 (linux)	115200



(a) Signal fra TX på ESP32, her sendes tallet 1



(b) Testing av seriekommunikasjon

Figure 8: Oppkobling av oscilloskop for å måle logiske nivåer og debugge pakkeformatet.

## 4.5 ESP32

Vi bruker to utviklingskort med ESP32-MCU. Disse kan programmeres over USB og kom med ferdigmontert OLED-fargeskjerm, noe som sparte oss for arbeid med å montere dette selv. De to ESP-ene har som oppgave å ta imot og videresende pakker som kommer fra **radio→seriell** og fra **seriell→radio**.

Vi har bevisst ikke gitt ESP-ene flere oppgaver for å holde de to arduinosketsjene likest mulig. For å kunne programmere de to ESP-ene med den samme kildefilen har vi benyttet C++ sin preprosessor (se figur 10).

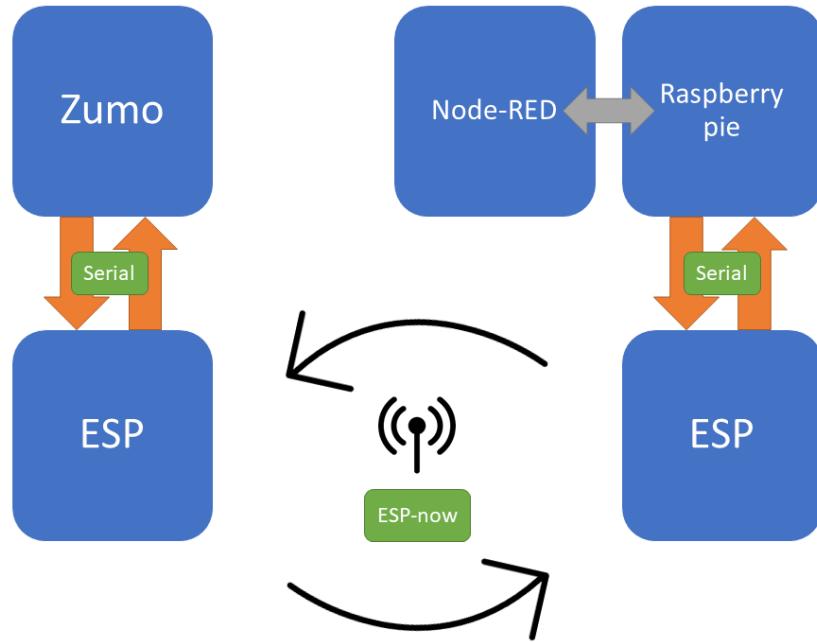


Figure 9: Oversikt over kommunikasjon.

```

1 #define IS_ZUMO      // Kommenter ut for    flashe RPi-ESP, behold den for    flashe ZUMO-ESP.
2
3 #ifndef IS_ZUMO
4 uint8_t PEER_MAC_ADDR[] = {0x84, 0xCC, 0xA8, 0x61, 0x51, 0x28}; // (ESP med rett resetknapp)
5 #else
6 uint8_t PEER_MAC_ADDR[] = {0x84, 0xCC, 0xA8, 0x61, 0x56, 0x88}; // (ESP med skeiv resetknapp)
7#endif

```

Figure 10: Eksempel på bruk av preprosessor for å preparere ESP-kildekodefila for kompilasjon.

#### 4.5.1 Esp Now

For trådløs kommunikasjon valgte vi å bruke **ESP Now**[7]. Dette er en radiokommunikasjonsprotokoll laget av Espressif som bruker 2,4GHz som bærebølge. ESP Now lar to eller flere ESP-er kommunisere direkte med hverandre uten å trenge å gå via en trådløs ruter, noe som gir oss fordelen at de to ESP-ene våre vil starte kommunikasjon nærmest øyeblikkelig når de skrus på.

Navn	MAC-addresse	oppgave
Raspberry-Esp	84 : CC : A8 : 61 : 56 : 88	bindeledd mellom RPi og ZUMO-Esp
Zumo-Esp	84 : CC : A8 : 61 : 51 : 28	bindeledd mellom ZUMO og Raspberry-Esp

Figure 11: Oversikt over oppgavene til de to ESP32-ene.

#### 4.5.2 TTGO OLED-fargeskjerm

Den ferdigmonterte OLED-skjermen har vi brukt for å printe informasjon om sendte og mottatte pakker, noe som har vært nyttig under utviklingsprosessen. Under implementering av *Esp Now* var det nyttig å kunne vise Esp Nows ferdigdefinerte feilmeldinger. Senere i prosjektet har vi gått over til å printe pakken som sendes frem og tilbake over seriell og radio i hexadesimal, noe som har blitt værende på det ferdige produktet. Dersom en pakke blir sendt over ESP Now uten å få tilbake bekreftelsesmelding (*ACK*) vises dette med rød skrift.

### 4.6 Ekspansjonskretskort for Zumo

For å sørge for god kontakt mellom Zumoen og den ene ESP32-en loddet vi sammen et eget kretskort. Dette kobler de fjorten pinnene som er eksponert på toppen av Zumoen til de relevante pinnene på ESP-en slik at den blir spenningsatt. De relevante pinnene på den ultrasoniske sensoren og servomotoren er også koblet til Zumoen via kretskortet. Alle hovedkomponentene (TTGO ESP32, servo, ultrasonisk sensor) samt ekspansjonskretskortet selv er avtakbare, dette er gjort vha. *pin-headers* eller med 2,5mm krimpekobling. Kommunikasjonskablene mellom Zumo og ESP går igjennom en TXS0108E logisk nivåskifter som oversetter aktivt mellom 3,3V og 5V-logikk. Vi brukte M2-bolter, muttere og mutterstag for å montere servoen på toppen av en tynn MDF-sponplate, som vi skar til og pussa slik at den ikke skjuler skjermen. Ledningene til den ultrasoniske sensoren loddet vi permanent på for å motvirke det mekaniske stresset fra servomotoren.

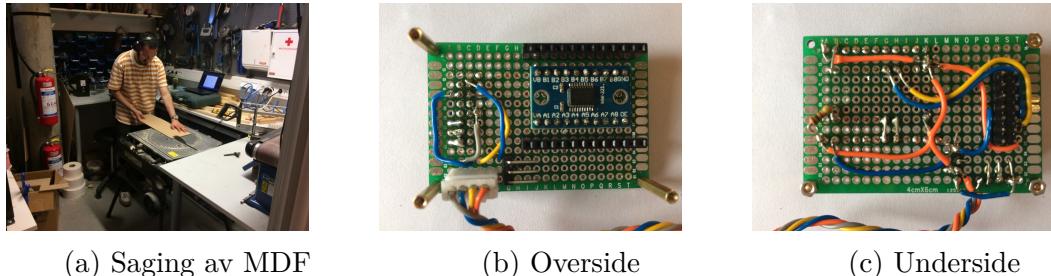


Figure 12: Tilvirkning av kretskort på verkstedet

### 4.7 Node-Red

For IOT-applikasjonene våre benytter vi oss av Node-RED[9]. Node-RED er et visuelt «flow basert» programmeringsverktøy som egner seg godt til å sy sammen flere komponenter og sensornoder til et system. En Raspberry Pi er koblet til wifi og kjører Node-RED. Siden vi valgte å bruke ESP-now til å kommunisere mellom ESP-ene våre, istedenfor

MQTT, må RPi-en både motta og sende data over seriellporten (/dev/ttyUSB0 på RPi-en). Pakkene med data som mottas via serial går gjennom en buffer parser for å sortere data og sende den videre individuelt til de aktuelle nodene i flowen. Ved hjelp av biblioteket node-red-dashboard kan man enkelt lage et brukergrensesnitt på nett, uten å sette opp en nettside selv med HTML. De ulike dashboard-nodene kan vise frem data på flere nyttige måter, eller sende outputs fra en bruker i et brukergrensesnitt. I tillegg til ferdige noder kan man lage sine egendefinerte funksjoner som gjør mer spesifikke oppgaver etter behov. Siden Node-RED er basert på node.js skrives disse funksjonene i JavaScript. Siden ESP-en som skal motta data fra Raspberry Pi er avhengig av å motta pakker med bestemt lengde og start/stopp bytes, bruker vi egendefinerte funksjoner til å legge verdiene fra PID-sliderne inn i slike pakker med data før pakken sendes videre.

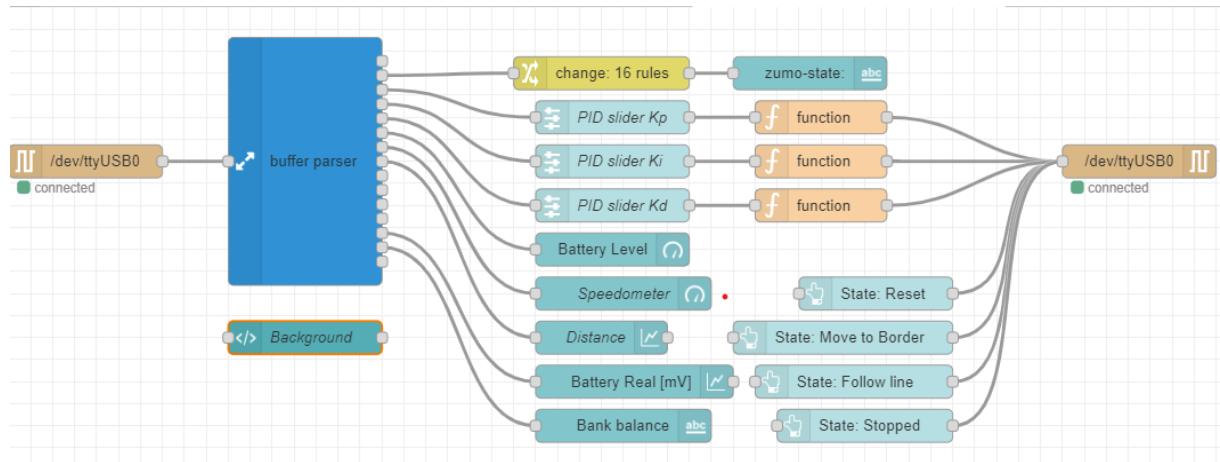


Figure 13: Node-Red-flowen som lager grensesnitt og behandler data på RPi

Node-RED dashbordet finner vi ved å koble oss på <http://gunnmy.likes-pie.com:1880> i en nettleser. I brukergrensesnittet vårt, eller dashbordet, har vi muligheten til å overvåke og styre Zumo-bilen. På dashbordet ser vi fart, software-batteriet, spenningen på zumoens faktiske batteri og avstanden registrert av den ultrasoniske sensoren. Man kan også styre zumoen til en viss grad ved å endre states på zumo-bilen. Statene man har å velge mellom da er RESET, MOVE\_TO\_BORDER, FOLLOW\_LINE og STOPPED. I tillegg kan man også endre på linjefølgerens PID-verdier ved å endre på tre slidere.

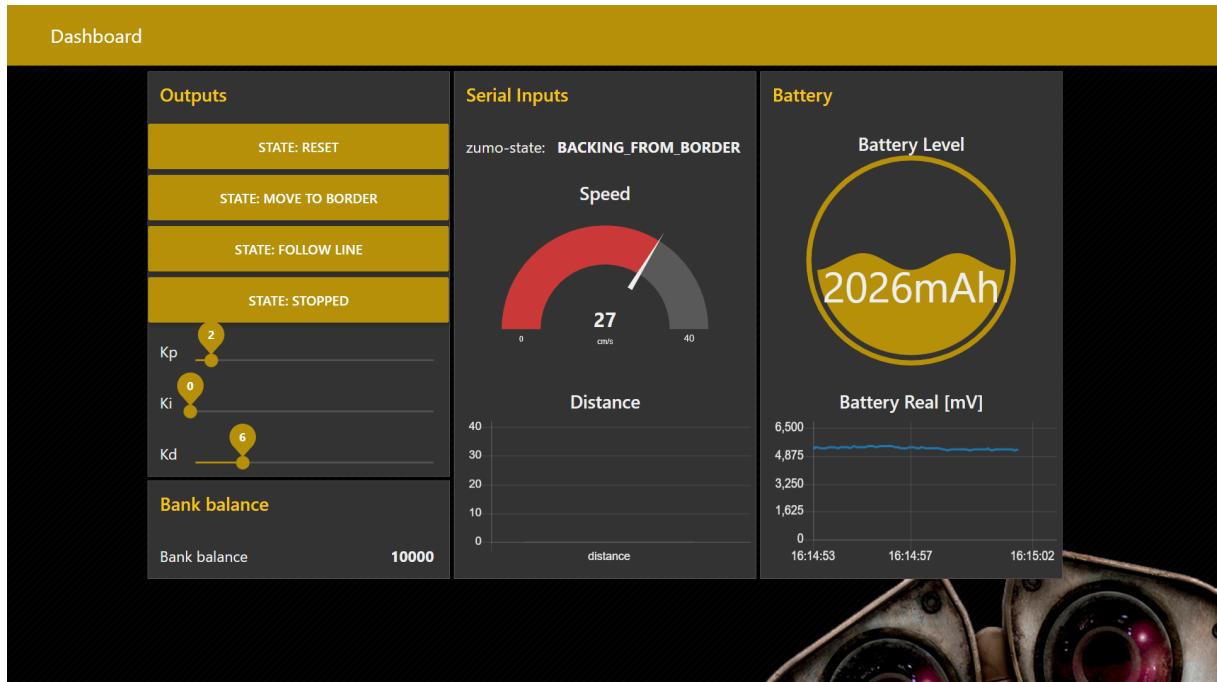
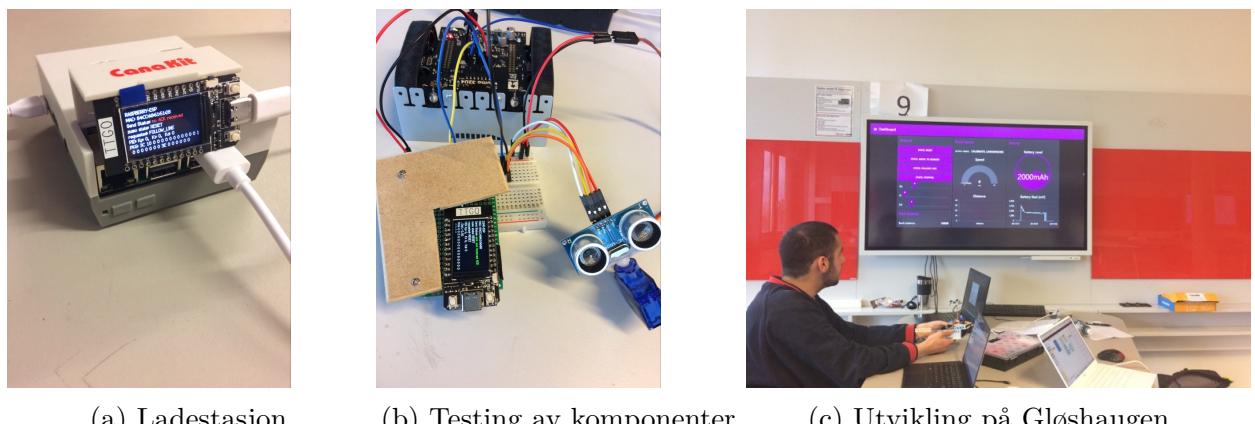


Figure 14: *Dashboard* som oppdateres i sanntid på nettet



## 5 Resultat

### 5.1 Generell virkemåte

For at Wall-E skal fungere trenger den en lukket bane av sort teip. I en ende av banen må den ha en «inngang», som fører til en blindvei som markeres med en bredere stripe med teip. Den brede teipen i enden av inngangen fungerer som lade/miljøstasjon. Inne på banen plasseres tomme brusbokser som Wall-e skal hente og kjøre til stasjonen.

Når Wall-E starter, vil den først kalibrere linjesensorene. Når det er gjort dyttes Wall-E inn på banen slik at den kan begynne å se etter bokser. Wall-E søker ved å kjøre rette linjer inne på banen, til den enten treffer kanten av banen, eller ser en boks med den ultrasoniske sensoren. Hvis den treffer kanten mer enn fem ganger på rad vil den snu seg langsomt i ring mens den ser etter bokser, før den tar opp kjøringen igjen. Når den har funnet en boks vil den kjøre mot boksen slik at den havner i plogen. Deretter kjører den i en rett linje til den møter kanten av banen. Så vil Wall-E, med boksen fremdeles i plogen, følge teipen til den kommer til stasjonen. Der vil den få betalt for arbeidet sitt, og kan bruke pengene sine til å lade software-batteriet sitt for en pris. Til slutt vil den rygge ut av stasjonen, slik at boksen blir liggende igjen. Wall-E er ute på banen og starter søkingen på nytt. Wall-E lager også ulike lyder når den finner bokser og leverer de i stasjonen.

I et brukergrensesnitt har vi full oversikt over hva Wall-E gjør til enhver tid. Man kan også endre ulike states og PID-variabler.

### 5.2 Resultat av test

Vi utførte fire tester der vi så hvor mange av boksene på banen zumo-bilen klarte å levere. Vi startet med seks bokser. Spenningen på batteriet ble lest av før og etter hver test. Vi tok også tiden og fulgte med om zumo-bilen holdt seg inne på banen, se tabell i figur 18.

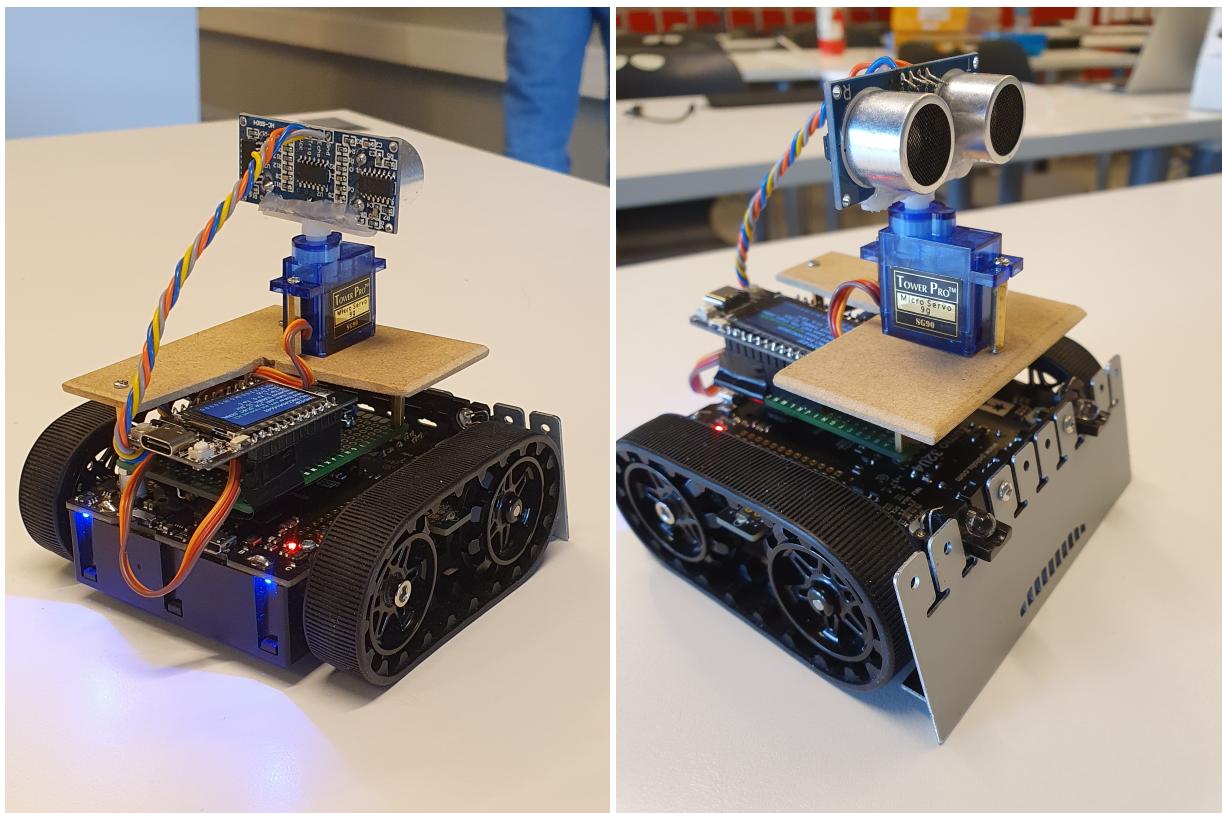


Figure 16: WALL.E, her avbildet før plogen ble montert

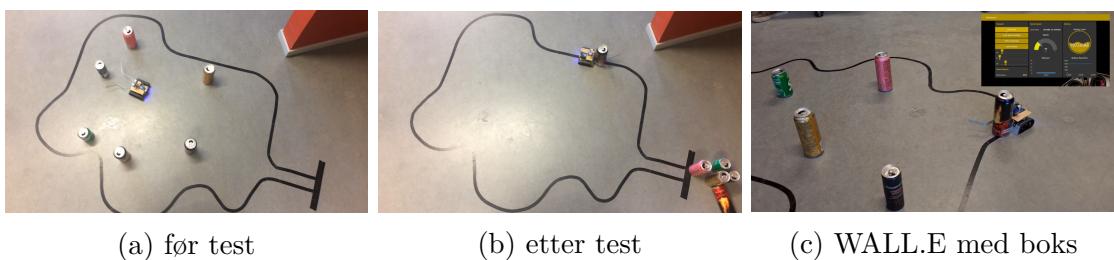


Figure 17: Under test tilsvarende de fra figur 18, boksene er tilfeldig plassert. I (a) og (b) klarte WALL.E å levere 6 av 6 bokser.

test-nr.	bokser totalt	bokser leveret	batterinivå [mV] før test	batterinivå [mV] etter test	holdt seg inne i banen	tid	leveringsprosent
1	6	5	5050	5010	ja	2:01	83%
2	6	5	5050	4990	ja	3:45	83%
3	6	3	5000	4990	ja	1:25	50%
4	6	5	4990	4980	ja	3:31	83%

Figure 18: Resultat av fire testkjøringer der boksene er plassert vilkårlig inne i innhegningen

## 6 Diskusjon

### 6.1 Teknologivalg

Noen av teknologivalgene vi gjorde kom til å forme utfallet av prosjektet, og de hadde alle både fordeler og ulemper med seg. Når et valg ble gjort tidlig i prosjektet ble det som regel med oss hele veien.

#### 6.1.1 Bruk av ESP Now i stedet for MQTT

Vi valgte å bruke ESP Now siden det hørtes ut som en god og forutsigbar metode å kommunisere mellom to ESP-er på. Hovedfordelen er at ESP-ene kommuniserer direkte med hverandre og slipper å gå via en ruter, og dette gjør at vi slipper å hardkode inn ip-addresser i kildekoden (vi hardkoder derimot MAC-addresser, som ikke endrer seg). Dette gjør også at ESP-ene vil kunne opprette kommunikasjon hvor som helst. Valget av ESP Now har også endret måten vi sender og håndterer data på, siden det i praksis tvang oss til å skrive en egen kommunikasjonsprotokoll. I stedet for å bruke *topics* endte vi opp med en monolittisk pakke på 21 byte. Dette er ikke optimalt mtp. å minimere overføringsmengde, men det er like mye eller mindre enn om vi hadde overført strenger som man typisk gjør med MQTT.

Når vi først fikk de to ESP-ene til å overføre pakka frem og tilbake fungerte det veldig bra, og vi trengte ikke endre på ESP-koden mer enn om vi la til flere bytes i *Packet.h*. Dette var veldig praktisk siden det lot oss fokusere på Zumo-koden og Node-Red. Det viser seg også å være veldig god rekkevidde på ESP Now, noe som ikke er så relevant for dette prosjektet men kan være praktisk for senere prosjekt.

### 6.2 Problemer vi har lært av

#### 6.2.1 Servomotor og pinouts fra ZUMO32u4

Servomotoren ("halsen" til WALL.E) fikk vi ikke til å fungere samtidig som motorene. Det viste seg at de fire ledige GPIO-pinnene som er eksponert på toppen av Zumoen allerede er i bruk til forskjellige interne zumo-funksjoner slik som indikatorlysdioder for RX og TX, og værst av alt – for den høyre beltemotoren. Det ble derfor vanskelig å sende PWM-signal for å styre servoen og samtidig sende/motta pakker eller kjøre motorene. Vi forsøkte forskjellige ting for å unngå dette problemet, blant annet å endre PWM-timer eller å dekonstruere servo-kontrolleren mellom hver gang servoen skulle styres. Ingen av disse løsningene var særlig elegante så vi bestemte oss til slutt for å ikke bruke servoen og heller la WALL.E snu hele kroppen for å søke etter bokser.

### 6.2.2 Wall-E gikk sporadisk i kalibreringsmodus

Når vi testet Wall-E's ulike funksjoner slet vi med at Zumoen sporadisk gikk inn i staten CALIBRATE\_LINESENSORS. Problemet viste seg å være at spenningen på zumoens batterier var for lavt. Bilen ville til tider skru seg av et lite øyeblikk, før den skrudde seg på igjen. Dette var nok til at zumoen startet på nytt og startet i kalibrerings staten. Bytte av batterier løste problemet.

## 6.3 Forbedringer

Vi har selv sagt måtte avgrense tidsbruken på prosjektet grunnet andre fag, og det er derfor en del ting vi gjerne skulle gjort men ikke fikk tid til.

### 6.3.1 Optimalisering av kommunikasjonsprotokollen

Kommunikasjonsprotokollen som er definert i *Packet.h* har noen åpenbare forbedringer. Det mest nærliggende ville vært å lage en form for pakketittel (*header*) som beskriver hva slags informasjon som følger i resten av pakka, og kun sende små pakker med de verdiene som til enhver tid oppdateres. Dette ville minsket mengden data som sendes fra 21 byte ned til omrent 5 siden omrent 70% av bytene i pakkene har verdien 0 slik som systemet funger nå. Vi kunne også pakket de boolske variablene sammen til én byte, men dette ble ikke gjort av hensyn til Node-Red siden binæroperasjoner i javascript viste seg å ta lengre tid enn det praktiske utbyttet var.

Det største problemet med kommunikasjonsprotokollen er likevel at de to byte som indikerer at pakka starter og stopper har verdier som vi også kan ønske å sende i pakka. Vi bruker ASCII-karakterene "<" som *start\_byte* og ">" som *stop\_byte*, som henholdsvis oversetter til verdiene 60 og 62 i desimal. Dette vil si at dersom en av byteene midt i pakka inneholder verdien 62 vil dette indikere at pakka er slutt. For å unngå dette umiddelbare problemet måtte vi anta at pakkas integritet var i hevd og kopiere over 21 bytes hver gang det kom inn en *start\_byte* og det ikke allerede var en overføring i gang. Selv om dette ikke skapte noen problem for vår monolittiske og uniforme pakke på 21 bytes ville det skapt problemer dersom vi ønsket å videreutvikle kommunikasjonsprotokollen til å også støtte variable pakkestørrelser og bedre pakkeintegritetssjekking. En mulig løsning kan være å begrense det gyldige tallområdet for hver byte til f.eks. 250 slik at de fem gjenværende bitkombinasjonene indikerer spesielle kommandoer slik som *start\_byte* og *stop\_byte*, men dette ville igjen skapt komplikasjoner for datatyper bestående av flere byter. Den beste løsningen på problemet ville nok vært å implementere en eksisterende datakommunikasjonsprotokoll som allerede tar hensyn til disse problemene, noe vi kanskje vil lære mer om neste semester for de som tar faget *datakommunikasjon*.

### 6.3.2 Direktestyring av zumoen med spillkontroller

Kommunikasjonssystemet vi har laga muliggjør direktestyring av zumoen, enten via Node-Red eller en spillkontroller. Dette var en idé vi hadde siden vi fikk utdelt to NES-handkontrollere med Raspberry Pi-en vår, men vi fikk ikke tid. Den enkleste måten å implementere dette på er å la NES-kontrollerens USB-pakker bli tolket av et python-script på Raspberry Pi-en, som igjen setter verdier inn i pakker som følger kommunikasjonsprotokollen vår. Dette lar seg gjøre med pythons *serial*- og *struct*-pakker, som vi allerede har erfaring med fra testing av seriellkommunikasjonen mellom RPi og RPi-ESP32. Python-scriptet kunne vi starta automatisk som en *systemd-service*, slik som vi allerede gjør med Node-Red.

### 6.3.3 Låsing av last under transport

Den originale planen var at WALL.E skulle holde på boksene under transporten, men vi endte med en enklere løsning der boksen fanges av en plog og farten senkes når vi dyster en boks tilbake til ladestasjonen. Denne løsningen fungerer omrent 70% av gangene uten at boksene kanter på utsiden av plogen, så om det er viktig at boksene samles på ett sted er dette et område vi kunne ha forbedret. En holdmekanisme kunne blitt lagd ved hjelp av én eller to servomotorer med gripearmer, som enten kunne 3D-printes eller bøyes til av patentbånd slik den nåværende plogen er konstruert. Dette ville kutta tiden til henteprosedyren drastisk siden zumoen nå kunne kjørt linjefølgingsprosedyren med full fart i motsetning til 50% som den gjør nå.

### 6.3.4 Flere ESP32 noder

Med oppsettet vårt er det fullt mulig å legge til flere ESP32 noder. ESP-en som snakker med Rasberry Pi kan bruke ESP now til å sende og motta data fra flere ESP-er samtidig. Man kunne for eksempel ha ett ESP32-node med sensorer som måler lyset i rommet og fått zumo-bilen til å stoppe om det blir for lite lys.

## 6.4 Relevans med tanke på FNs bærekraftsmål

Gjennom dette prosjektet er vi innom flere temaer som er relevante for FNs bærekraftsmål. I vårt prosjekts «smart city» går Wall-E og rydder tomme brusbokser ut av byen til en miljøstasjon. Et slik system i en by vil kanskje gjøre at flere bokser blir resirkulert enn om man ikke har det. FNs bærekraftsmål nr. 12 som omhandler ansvarlig forbruk og produksjon, sier at for å sikre gode levekår for nåværende og fremtidige generasjoner er vi nødt til å redusere ressursbruken [10]. Et system som automatisk plukker opp resirkulerbart søppel og leverer det til en miljøstasjon vil helt klart kunne redusere ressursbruken i et samfunn ved at mer av søpla faktisk blir resirkulert, og ikke havner i naturen. Når

mer av søpla forsvinner vil innbyggerne i denne byen også kunne oppleve bedre helse og livskvalitet, noe FN nevner som sitt bærekraftsmål nr. 3.

## 7 Konklusjon

Til tross for at vi kun er tre personer på gruppa har vi fått til å skape et fungerende system som henter bokser. Noen funksjoner som vi ville ønsket oss har vi ikke fått med, men likevel har vi nok til å skape den funksjonaliteten vi så for oss. Spesielt ønsket vi oss en roterende ultrasonisk sensor ved hjelp av en servomotor, lik en radar. Zumo-bilen har et fungerende software-batteri som lades i en ladestasjon, den har en linjefølger som er enkel å justere fra brukergrensesnittet, og den bruker zumoen selv som sensornode der kommunikasjonsprotokollen ikke setter grenser for hvor mange sensorer vi kunne montert. I tillegg kan man ha full oversikt over hva som skjer til enhver tid ved bruk av et brukergrensesnitt fra Node-RED. Man kan også bytte mellom 4 utvalgte tilstander man ønsker kjøre på zumo-bilen. Kommunikasjonen mellom ESP-ene har vi løst ved bruk av ESP now, som har vist seg robust og effektivt. Flere tester av hele systemet viser at programmet kjører som det skal, uten intervensjon av brukeren.

Vi har hatt jevnlig arbeidsøkter på campus der vi har jobba jevnt og trutt, og når det ble naturlig fant vi oss arbeidsområder der vi ville fokusere mer på personlig. Håkon tok ansvar for Node-Red, Gunnar jobba mest med serie- og radiokommunikasjonsbiten mens André jobba mye med systemene på Zumoen. På arbeidsøktene ble det derfor mer tid til å forsøke å integrere alle systemene sammen, og å skrive Zumoen hovedprosedyre.

## References

- [1] Link til videopresentasjonen:  
[https://studntnu-my.sharepoint.com/:f/g/personal/haaknord\\_ntnu\\_no/Eo4F-WGHfNIsuowb65ctLsBELsqa7RY26vWkMFRQD8zbw?e=fJRU6f](https://studntnu-my.sharepoint.com/:f/g/personal/haaknord_ntnu_no/Eo4F-WGHfNIsuowb65ctLsBELsqa7RY26vWkMFRQD8zbw?e=fJRU6f)
- [2] Arduinos offisielle nettside med dokumentasjon <https://www.arduino.cc/>
- [3] Datablad for HCSR04 ultrasonisk sensor <https://www.electroschematics.com/wp-content/uploads/2013/07/HCSR04-datasheet-version-1.pdf>
- [4] Brukerveiledning for Zumo32u4 [https://www.pololu.com/docs/pdf/0J63/zumo32u4\\_robott.pdf](https://www.pololu.com/docs/pdf/0J63/zumo32u4_robott.pdf)
- [5] Dokumentasjon for Zumo32u4 sitt programvarebibliotek  
<https://pololu.github.io/zumo-32u4-arduino-library/index.html>
- [6] Datablad for TXS0108E <https://www.ti.com/lit/ds/symlink/txs0108e.pdf>
- [7] Dokumentasjon for ESP-NOW <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/espnow.html>
- [8] Datablad for SG90 servomotor [http://www.ee.ic.ac.uk/pcheung/teaching/DE1\\_E/stores/sg90datasheet.pdf](http://www.ee.ic.ac.uk/pcheung/teaching/DE1_E/stores/sg90datasheet.pdf)
- [9] Dokumentasjon for Node-RED <https://nodered.org/docs/api/>
- [10] FN, "FNs bærekraftsmål", (2022, mai 2), hentet fra <https://www.fn.no/om-fn/fns-baerekraftsmaal>
- [11] Kildekode for prosjektet: [github.com/andrejvi/WALL\\_E](https://github.com/andrejvi/WALL_E)

## Vedlegg

1. Vedlegg 1 Bidragserklæring.pdf på [OneDrive](#)
2. Vedlegg 2 Videopresentasjon på [OneDrive](#)
3. All kode ligger på [github](#)