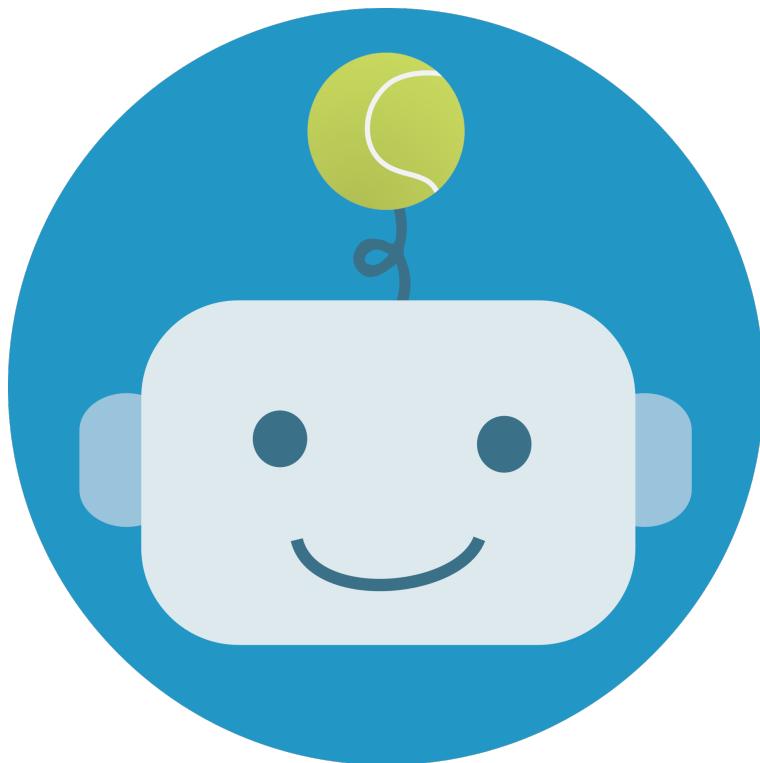


IROB Documentatie

Joey de Ruiter & Sergi Philipsen

10 juli 2020



*Hoe de “Tennis Ball Bot” tot stand is gekomen
Voor de minor: “Robotica, Domotica en Industriële automatisering” te Hogeschool Leiden*

Voorwoord

“Al sinds kinds af aan vond ik het leuk om apparatuur uit elkaar te halen. Oude harddisks en versterkers werden niet gespaard. Ik vond het fascineren om alle kleine onderdeeltjes samen te zien werken als één geheel. Later ben ik nog met elektronica blijven hobbyen, helaas niet zo veel als ik zou willen. Naast een microfoon die wat gesoldeerd moest worden en een scherm van een telefoon die vervangen moest worden, heb ik het nooit echt aangedurfd om iets technischere dingen te doen. Ik wilde hier graag meer van leren en mijn enthousiasme voor elektronica een nut geven. Om deze reden ben ik de minor gaan doen. Ik kijk er naar uit om binnenkort een oude VU-meter en Macintosh SE werkend te krijgen!” – Sergi Philipsen

“Mij leek de minor interessant omdat het te maken had met het schrijven en ontwikkelen van embedded software. Tevens was het ook een groot plus punt dat de eindopdracht van de minor een zelfgekozen opdracht zou zijn.” – Joey de Ruiter

Allereerst willen we Vincent Bakker en Herman van Haagen bedanken voor de lessen en begeleidingen die ze ons gegeven hebben. We kunnen ons voorstellen dat het als docent erg lastig is om kennis over te dragen op het moment dat de leerlingen geen directe feedback in de klas kunnen geven. Ondanks het werken vanaf thuis denken we dat het ze goed gelukt is om ons nieuwe dingen te leren en ons rijker van kennis te maken. Hiernaast willen we ook Justin Verkade bedanken voor het repareren van een elektromotor en Timo Kleverlaan voor het printen van een 3D model. Zonder hun hadden we niet tot dit mooie resultaat kunnen komen.

Inhoudsopgave

1 Inleiding	4
2 Voor de sprint	5
2.1 Het idee	5
2.2 Onderzoek	5
3 Sprint 1	6
3.1 Ram uitwerken en ontwerpen	6
3.2 Tennis ball model trainen	7
3.3 Web-applicatie bouwen voor het instellen en lezen van de positie van de robot	9
3.4 Ballen detecteren en positie verkrijgen	10
3.5 Bepaal welke bal het dichtstbij is door de breedte uit te rekenen	11
3.6 Backend voor de web-applicatie maken	11
3.7 Verbinding tussen Raspberry Pi en Arduino	12
3.8 Lokale positie berekenen door middel van BLE	13
4 Sprint 2	17
4.1 Serveer de web-app via Flask	17
4.2 Op de gedetecteerde ballen afrijden	17
4.3 Ballen naar het verzamelpunt brengen	20
4.4 Logica voor het verzamelpunt, zones en modus maken	20
4.5 Navigatie klasse maken	21
4.6 Binnen de zones blijven	21
4.7 Obstakels ontwijken	21
5 Reflectie	22
5.1 Wat ging goed	22
5.2 Wat kon beter	23
A Code & Middelen	25
B Ontwerp	25
C Showcase	25
D Onderzoek	25
Referenties	42

1 Inleiding

Dit project is gemaakt als eind-opdracht van de Minor: “Robotica, Domotica en Industriële automatisering” te Hogeschool Leiden. In dit verslag is vast gelegd wat we in de loop van het project hebben gemaakt en meegemaakt. Zowel bij dingen die goed als minder goed gingen staat vastgelegd hoe we dit hebben gedaan. Ook is er aangegeven hoe we het beter zouden doen in de reflectie.

Dit project hebben we uitgevoerd met Scrum in gedachte. We zouden werken op basis van sprints om het product zo snel mogelijk te realiseren en taken overzichtelijk te houden.

Het project is opgedeeld in vier delen en twee sprints:

1. Het idee: Bedenken wat we willen realiseren in het project.
2. Het onderzoek: Uitwerken wat we gaan maken en hoe.
3. Iteratief ontwikkelen in sprints:
 - (a) Sprint 1: Alle grote onderdelen van het project werken in brede lijnen.
 - (b) Sprint 2: Alle onderdelen worden bij elkaar gevoegd en afgesteld.
4. Reflectie: Documenteren wat goed en fout ging.

De sprints duren ongeveer elk twee weken. Voorgaand hebben we een onderzoek uitgevoerd om duidelijk te krijgen wat het was, dat we gingen maken.

2 Voor de sprint

2.1 Het idee

Aan het begin van het project hadden we meerdere ideeën verzonnen. Zo hadden we verzonnen om een soort op afstand bestuurbare battle-bot te maken, die door middel van een livestream kon laten zien waar die was. Een ander idee was om een prullenbak naar je toe te laten rijden door met object detectie een handgebaar te detecteren en hier op af te komen. Echter hadden deze ideeën wat problemen en hebben we voor een middenweg gekozen: Automatisch tennisballen verzamelen op een tennisveld met afstandsbediening voor extra besturing.

2.2 Onderzoek

Voorafgaand aan de sprint's hebben we een onderzoek uitgevoerd. Deze is uitgevoerd om er echter te komen *wat* we gingen maken en *hoe*.

We hebben het onderzoek gestart met een "Probleemanalyse" met daarin de "Aanleiding en context", "Probleemstelling" en "Doelstelling". Dit gaf ons een goed beeld over waarom we het onderzoek deden. In "Onderzoeksopzet & Uitvoering" hebben we een hoofdvraag en deelvragen opgesteld waarvan we vonden dat deze ons dichter bij het gewenste eindresultaat zou brengen. De hoofdvraag luid: "*Wat is de beste manier om een robot efficiënt en snel tennisballen op te laten rapen?*" De deelvragen zijn er op gericht om stapje voor stapje een beter antwoord op de hoofdvraag te krijgen. De vragen waren erg breed gesteld zodat we veel mogelijkheden open hielden. Dit vonden we belangrijk omdat we niet goed wisten in wat voor situatie de robot het beste gebruikt zo kunnen worden en hoe deze te werk zou gaan. Enige termen die onbekend zijn voor een leek werden in het "Theoretisch kader" toegelicht.

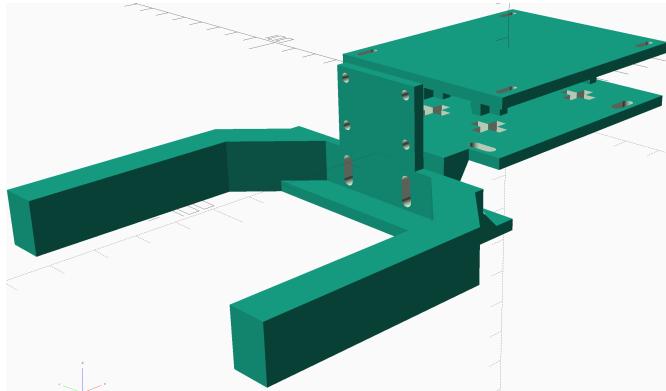
In het hoofdstuk "Resultaten" hebben we alle mogelijke antwoorden – die we konden bedenken – op de vragen gegeven. Ook hebben we de vragen op het internet opgezocht en de bronnen bij de resultaten gemeld. Het maakte ons niet heel veel uit of de resultaten die we vonden haalbaar of praktisch waren, aangezien ze ons wel een beter beeld zouden kunnen geven voor de aanpak in het algemeen. De antwoorden die we bij de "Resultaten" hadden gevonden, hebben we vervolgens in de "Onderzoek & discussie" besproken en beargumenteerd wat we de beste oplossingen vonden. Dit deden we door middel van bevindingen en eerdere ervaringen. Ook hebben we proeven gedaan om zeker te weten welke techniek goed en bruikbaar zou zijn. In de "Conclusie" vatten we samen wat we in de "Onderzoek & discussie" hebben geconcludeerd. Ook is er uit op te nemen wat we zullen gaan doen om het product te realiseren.

Het onderzoek en de conclusie hiervan zitten als bijlage bij dit document.

3 Sprint 1

3.1 Ram uitwerken en ontwerpen

Om ons doel te bereiken om ballen te verzamelen en naar een zelf geselecteerde hoek te brengen hebben wij een enkel paar aanpassingen toepassen aan de buitenkant van de Zumo [12].



Figuur 1: Zumo mount

Een van deze aanpassingen was een grote vork aan de voorkant plaatsen, de binnenkant van deze vork spanpt 7.5cm wijdt, dit is ongeveer 1 cm groter dan de officiële diameter van een tennisbal (6,54cm - 68,6cm). Door deze marge toe te passen was het iets gemakkelijker om de tennisballen op te rapen. Tevens zit er een camerahouder aan de voorkant van de vork gemonteerd.

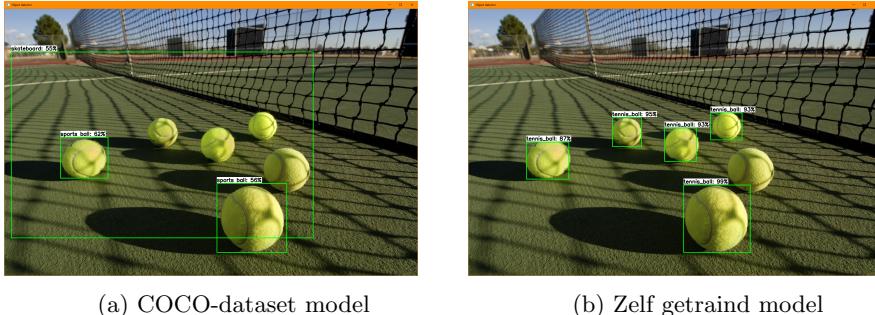
Een andere aanpassing die we hebben toegevoegd is een shield voor de Arduino Uno [1]. In dit shield zijn 4 plusjes uitgesneden waarmee we de houder voor de Raspberry Pi [9] in kunnen zetten. Dit kan gedaan worden zonder dat er schroefjes zijn die de onderkant van de Raspberry Pi raakt en mogelijk kortsluiting laat ontstaan. Verder hebben we aan de voorkant van het shield een extra montage mogelijkheid voor de vork geplaatst.

Als laatste aanpassing hebben we een Raspberry Pi houder toegevoegd, deze houder kan je monteren door middel van 4 plusjes die aan de onderkant van de houder zitten het shield van de Arduino in te duwen. De Raspberry Pi zelf kan monteerd worden door schroefjes door het bord te doen en deze door de gleuven van de houder vast te monteren.

3.2 Tennis ball model trainen

Om te testen of we een eigen model beter zou zijn, hebben we eerst een eigen model getraind en deze getest tegen de COCO-dataset [3] model.

Voor het trainen van het model had Sergi grofweg 500 afbeeldingen van tennisballen gedownload en gelabeld. Zoals te zien in figuur 2, werkte de detectie net wat beter dan die van de COCO-dataset.



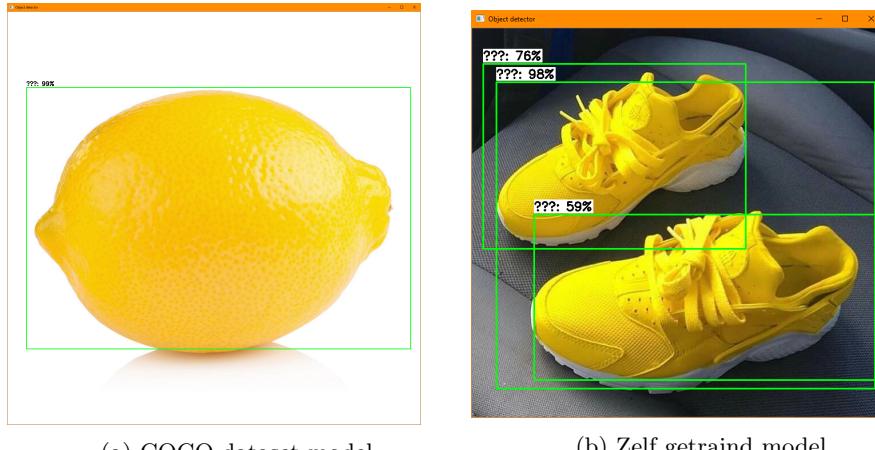
Figuur 2: Het verschil tussen het voorgetraind en het zelf getrainde model.

Er is er bij 2b te zien dat niet alle ballen worden gedetecteerd, maar dat is in ieder geval beter dan het skateboard dat bij 2a zou worden gedetecteerd. We zijn toen tot de conclusie gekomen dat een zelf getraind model beter was, aangezien er dan geen verwarring plaats kan vinden. Echter bleek wel dat het zelfgetrainde model meer moeite had met verschillende belichting. Er is toen voor gekozen om het dubbele aantal afbeeldingen te trainen. Dit zorgde er voor dat de detectie wat beter werkte en elke bal snel detecteerde. De focus lag dan ook vooral op ballen die ver weg – en daardoor niet even scherp – waren. In figuur 3 is een slecht belichte foto op 10 meter afstand genomen. De detectie is niet perfect, maar werkt nog redelijk goed.



Figuur 3: Een bal op 10 meter afstand, in een slecht belichte gang.

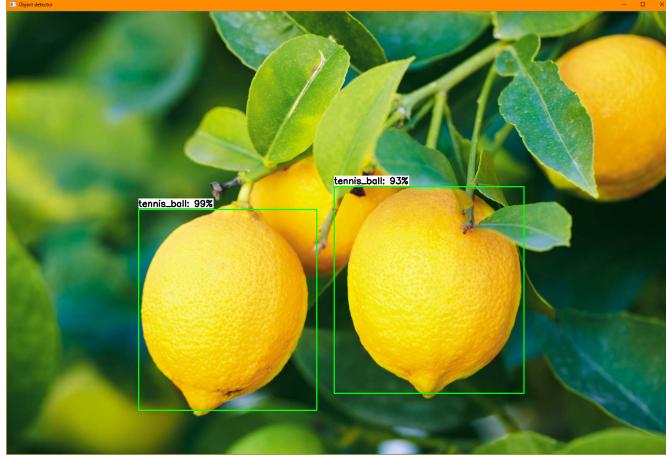
Er is veel getraind op afbeeldingen die “out of focus” waren. Echter hadden we niet door dat hierdoor het model vrijwel alles wat geel was zou detecteren als een tennisbal. Om dit tegen te gaan zijn we gebruik gaan maken van hard-negatives. Dit zijn objecten die je expliciet markeert als niet-juist. Dit zou er voor moeten zorgen dat er tijdens het trainen meer rekening wordt gehouden met de ronde vorm van een tennisbal. In figuur 4 is te zien dat de objecten niet als tennisballen worden gedetecteerd.



Figuur 4: Objecten die door de hard-negative training niet als tennisballen worden gezien.

Waar ik achteraf achter kwam was dat de hard-negatives weer iets te veel op

de witte achtergrond hadden getraind. Dit resulteerde er in dat objecten die op tennisballen leken met een groene achtergrond, alsmede als tennisballen werden gedetecteerd. In Figuur 5 is hier een voorbeeld van te zien.



Figuur 5: False positive

3.3 Web-applicatie bouwen voor het instellen en lezen van de positie van de robot

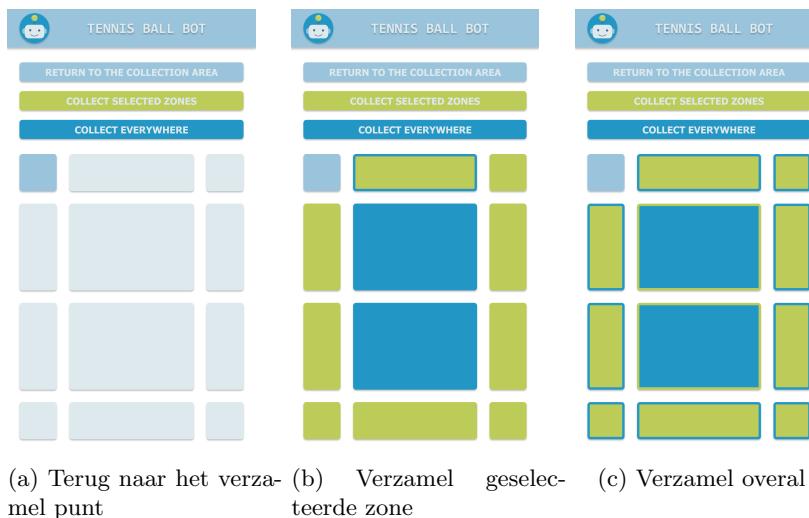
Voor de web-applicatie was eerst een design gemaakt in Figma. Dit is gedaan omdat we vooraf niet wisten wat de beste manier zou zijn geweest om de robot te bedienen. Uiteindelijke zijn we op een ontwerp gekomen waarin de gebruiker kan kiezen tussen drie verschillende modus:

1. Terug naar het verzamel punt: Dit kan worden gebruikt als de tennisser wil spelen en de robot niet in de weg wil hebben. De robot zal niet verzamelen.
2. Verzamel geselecteerde zone: De robot zal de ballen verzamelen in de geselecteerde zone. Dit is handig als er veel ballen achter de achterlijn liggen na een training.
3. Verzamel overal: Deze modus kan worden gebruikt na afloop van het trainen. De robot zal dan rustig over heel het veld gaan om de ballen te verzamelen.

Onder de drie knoppen voor de modus is een raster afgebeeld die grofweg een tennisveld representeert (er wordt geen rekening gehouden met dubbele lijnen). Het idee hiervan is dat de zones en het verzamelpunt kunnen worden aangegeven.

De uiteindelijke implementatie is geschreven in React [10], hier is voor gekozen, omdat we een goed werkende web-applicatie wilde, die niet al te veel boilerplate code had, zodat ontwikkelen sneller ging.

React heeft er uiteindelijk voor gezorgd dat het gebruik van de applicatie heel soepel verloopt. Zo kan er alleen een zone worden geselecteerd als de verzamelmodus aanstaat. Wel kan er altijd een verzamelpunt worden ingesteld. Op deze manier kan de robot in mode 1 alsnog van hoek naar hoek reizen. De manier waarop de huidige modus wordt aangegeven is door de velden op het raster aan te passen in hoe ze er uit zien en werken. Als de robot terug naar het verzamelpunt moet, worden alle zones een doffe kleur en zijn de knoppen disabled. In de verzamelmodus zal met een rand worden aangegeven welke zone wordt verzameld. In Figuur 6 zijn de drie verschillende modus te zien.



(a) Terug naar het verzamelpunt (b) Verzamel geselecteerde zone (c) Verzamel overal

Figuur 6: De drie verschillende modus van de web-app.

3.4 Ballen detecteren en positie verkrijgen

Voor het detecteren van de ballen wordt gebruik gemaakt van OpenCV [8] en Tensorflow Lite [11] in combinatie met de Google Coral [6]. Het script dat we hebben gemaakt is voor een groot gedeelte gebaseerd op het script van EdjeElectronics [4]. Hij heeft een tutorial gemaakt voor het maken van Tensorflow Lite modellen. Het script is aangepast om geen visuele output te geven. Dit maakt de detectie al wat sneller. Daarnaast zijn er wat instellingen voor de video opname aangepast, zodat dit voor meer frames kan zorgen. Hiernaast hebben we alles omgezet in OOP. Dit maakte het vervolgens wat makkelijker om het script in de rest van de applicatie te integreren. Ook hebben we een pauzeer functie toegevoegd die de detectie kan pauzeren. De resultaten van de object detectie worden in een detectie object gezet. Dit object rekent de positie

en afstand uit en zet deze in variabelen. De manier waarop dat gedaan wordt, is door de twee hoeken van het vierkant te pakken en van elkaar af te trekken, waardoor de breedte over blijft. Als deze breedte dan voor de helft van het meest rechtse punt wordt afgetrokken, blijft de positie van het midden van de detectie over. De functie die dit doet is te zien in figuur 7.

```
def get_width_and_position(boxes):
    xmin = boxes[1]
    xmax = boxes[3]

    width = (xmax - xmin)
    position = xmin + (width / 2)

    return width, position
```

Figuur 7: Functie om de breedte en positie van een detectie te bepalen.

3.5 Bepaal welke bal het dichtstbij is door de breedte uit te rekenen

Om uit de detectie de bal terug te krijgen die het meest dichtbij is wordt gebruik gemaakt van de breedte die voorheen is berekent. Je zou namelijk kunnen zeggen dat een bal die dichterbij ligt, er breder uit ziet op het beeld. In python zit een functie die het erg makkelijk maakt om de grootste waarde terug te krijgen. De functie die de juiste detectie terug geeft is erg Pythonic geschreven en is te zien in figuur 8.

```
def get_nearest_detection(detections):
    detection = max(detections, key=lambda d: d.width)
    return detection
```

Figuur 8: Functie om de dichtbije detectie te krijgen.

3.6 Backend voor de web-applicatie maken

De backend van de web-applicatie wilde we zo simpel en snel mogelijk maken. Om voor makkelijke integratie te zorgen hebben we er voor gekozen mo de web-server met Flask [5] te maken. Op deze manier hoeft er maar een programma gedraaid te worden op de Raspberry Pi. De Flask webserver beschikt over drie variabelen:

1. zone: De zone waarin verzameld wordt.

2. collection: Het verzamelpunt.
3. collection_mode: De verzamelmodus van de robot.

Voor elk van deze variabelen is een endpoint die een GET of POST request verwacht. Aan de hand van de soort request zal de webserver de variabele terug geven of opslaan. In figuur 9 is te zien hoe zo'n endpoint er uit ziet.

```
@app.route(ZONE_URI, methods=['GET', 'POST'])
def handle_zone():
    global zone
    if is_post(request):
        zone = request.json['zone']
    return json.dumps(zone)
```

Figuur 9: Een Flask endpoint die variabelen kan zetten en teruggeven.

3.7 Verbinding tussen Raspberry Pi en Arduino

De verbinding tussen de Raspberry Pi en de Arduino wordt in stand gebracht door middel van een seriële (UART) verbinding op de baudrate 115200. In het onderzoek hadden we beschreven dat we I2C zouden gebruiken. Alleen daar zijn we op het laatste moment van af geweken doordat het weinig extra functionaliteit meer bood in onze use-case en het complexer was om op te zetten op de Raspberry Pi.

Voor de seriële verbinding van de Pi wordt er gebruik gemaakt van de library “pyserial”. Deze library maakt het mogelijk om gemakkelijk naar de seriële buffer van de Arduino te schrijven. Voor de rest hadden we een kleine “service” gemaakt in het Python script die gebruik maakte van een lock. Door het gebruik van een lock was het niet meer mogelijk om met meerdere threads naar dezelfde buffer te schrijven, hierdoor was het niet meer mogelijk dat er een race-condition ontstond aan de Python kant.

Voor de seriële verbinding van de Arduino wordt de standaard libary gebruikt die Arduino automatisch meegeeft in Arduino projecten. Iedere “tick” van de Zumo wordt er gekeken of er iets in geschreven is. Zo ja, kijk of het een commando is en voer deze uit.

Iedere opdracht die opgestuurd wordt naar de Zumo moet zich houden aan de voor gespecificeerde notatie. De notatie ziet er zo uit: “commando=argument;”. Waarbij het commando de naam moet zijn van de opdracht die je wilt doen het argument een waarde moet zijn die de werking van de opdracht beïnvloed. Alle mogelijke commando’s staan in de tabel hieronder beschreven.

Commando	Argument	Beschrijving
move	[-400, 400]	Beweegt beide tracks dezelfde richting in.
left	[-400, 400]	Beweegt 1 track de gewenste richting in.
right	[-400, 400]	Beweegt 1 track de gewenste richting in.
center-left	[-400, 400]	Beweegt linksom rond z'n eigen as.
center-right	[-400, 400]	Beweegt rechtsom rond z'n eigen as.
honk	-	Maakt een “honk” geluid.
stop	-	Stopt beide tracks.

Figuur 10: Commandos van de Zumo

3.8 Lokale positie berekenen door middel van BLE

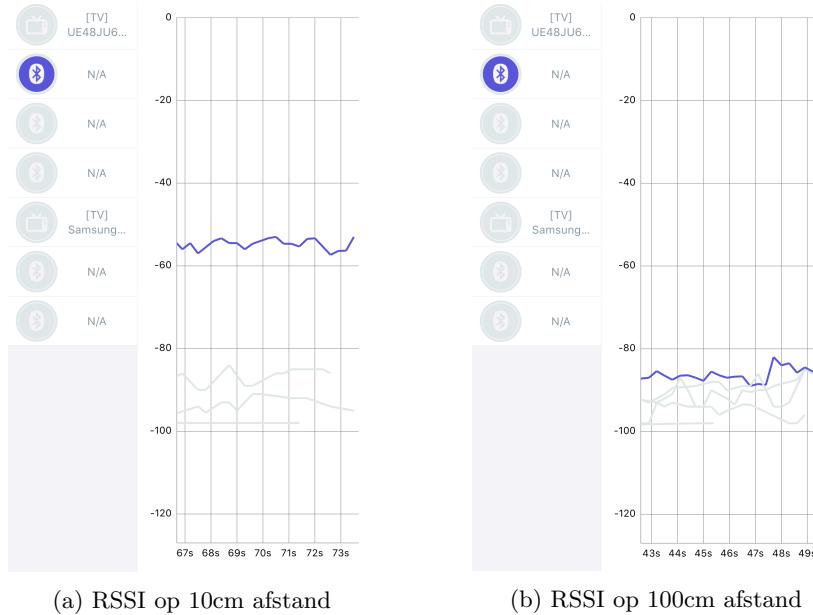
Om de lokale positie te berekenen van de TennisBallBot wordt er gebruik gemaakt van beacons. Deze beacons ondersteunen het iBeacon protocol [7] waarmee het mogelijk is de afstand te meten tussen de bot en de beacons.



Figuur 11: Beacon die aan staat

3

De afstand tussen de robot en de beacons wordt berekent door middel van RSSI (Recieved Signal Strength Indicator). Dit is een geschatte meting van hoe goed een apparaat een signaal ontvangt. De signaalwaarde wordt bij RSSI gemeten in decibel van 0 tot -120. Hoe dichter bij de 0 hoe sterker het signaal zou zijn.



Figuur 12: RSSI metingen op verschillende afstanden.

Na wat testen kwamen wij er achter dat de RSSI van ons BLE signaal ongeveer in de buurt kwam van andere apparaten die een passief Bluetooth signaal uitzenden. Onze test resultaten waren in de buurt van -70db op ongeveer 5cm afstand tot -80db bij 1 meter, na ongeveer 3 meter was het signaal niet meer bruikbaar door te veel ruis.

Het zou eventueel mogelijk zijn om de signaalsterkte van de beacons te versterken door onder andere een betere antenne op de robot en/of beacons te zetten, of een sterker transmitter/ontvanger op de apparaten te plakken.

Als protocol voor de beacons hadden wij het iBeacon protocol geïmplementeerd, de hoofdredenen hiervoor is omdat het een gestandaardiseerd protocol is met de al ingebouwde functionaliteit om afstand te berekenen op basis van RSSI. De code die hier voor nodig is hieronder te zien.

```

void create_beacon() {
    BLEBeacon beacon = BLEBeacon();
    beacon.setManufacturerId(0x4C00); // Fake Apple manufacturer id
    beacon.setProximityUUID(BLEUUID(resource_uuid));
    beacon.setMajor(resource_major);
    beacon.setMinor(resource_minor);
    beacon.setSignalPower(0xC1);

    BLEAdvertisementData advertisement_data = BLEAdvertisementData();
    BLEAdvertisementData scan_response_data = BLEAdvertisementData();

    advertisement_data.setFlags(0x04);

    std::string service_data = "";

    service_data += (char) 0x1A; // Len
    service_data += (char) 0xFF; // Type
    service_data += beacon.getData();

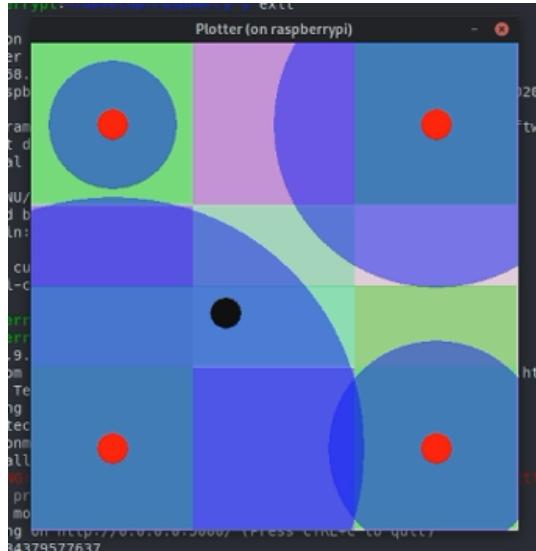
    advertisement_data.addData(service_data);

    ble_advertising->setAdvertisementData(advertisement_data);
    ble_advertising->setScanResponseData(scan_response_data);
}

```

Figuur 13: iBeacon implementatie op de beacons

Doordat we een gestandaardiseerd protocol hadden gebruikt voor de beacons was het mogelijk om een library te gebruiken voor het uitlezen van de beacons. De naam van deze Python3 library is “beacontools” [2] en maakt het mogelijk om gemakkelijk de advertise signalen van de beacons op te vangen en te verwerken. Met de data die wij ontvingen van de beacons hadden wij een simpele virtualisatie gemaakt. Deze virtualisatie toonde aan waar de robot dacht waar die op dat moment zou zijn op het veld.



Figuur 14: Virtualisatie van de positie van de robot

Een nadeel waar we achter kwamen was dat de enkele waardes van de beacon niet super accuraat waren, om een accurate positie te krijgen moesten je meerdere metingen hebben van dezelfde locatie. Hierdoor moesten we af en toe 20 seconden wachten om genoeg waardes te krijgen van huidige locatie. Verder was het lastig om de positie te updaten als de robot aan het rijden was. Dit kwam omdat het niet echt mogelijk was om van een bewegende positie genoeg data te krijgen.

De uiteindelijke nauwkeurigheid die wij konden uitlezen van de robot zat in de buurt van de 20cm tot 5cm. Met hoe dichter de robot in het midden was hoe beter de nauwkeurigheid werd van de positie.

4 Sprint 2

4.1 Serveer de web-app via Flask

Om te voorkomen om nog een applicatie zoals Nginx of Apache op de Raspberry pi te moeten installeren, configureren en draaien voor het serveren van de webapplicatie, leek het ons een goed idee om deze via Flask bereikbaar te maken. Dit wat helemaal niet lastig en hebben we op een vrij simpel maar toch elegante manier opgelost. Als de route “/” wordt aangevraagd, zal de index.html worden teruggegeven. Als er iets anders wordt opgevraagd, zal er worden gekeken of dit bestand bestaat op de Raspberry Pi. Als dit niet het geval is zal de webserver een 404 error terug geven. In figuur 15 is de code te zien.

4.2 Op de gedetecteerde ballen afrijden

Voor het afrijden op de bal hebben we een extra klasse gemaakt die gebruik maakt van de object detectie klasse de “detector”. Met behulp van een callback die de detectie terugkrijgt kunnen er acties worden ondernomen op basis van de positie van de bal. Allereerst leek het ons een goed idee om met een PID-controller op de bal af te rijden en bij te sturen waar dit nodig was. Echter ging de PID-controller een beetje gek doen op verschillende afstanden tot de bal. Als de Robot redelijk dichtbij was en werd aan gezet, zorgde de PID-controller er voor dat de robot een zwaai naar links deed en het doelwit ver voorbij ging. De robot was dan de detectie kwijt of vond een andere tennisbal. Om de PID-controller accurater te krijgen hebben we geprobeerd de PID waardes aan te passen en zo de controller wat bij te stellen. Dit duurde erg lang omdat de robot steeds verplaatst moest worden om dit opnieuw te kunnen testen. Omdat we – Joey en Sergi – niet dicht bij elkaar zijn, maakt dit het wel erg lastig en tijdrovend. We zijn van mening dat we dit op school veel beter hadden kunnen bijstellen. Echter moesten we nu met een simpele en snelle oplossing komen. Daarom zijn we richting een formule gedaan die aan de hand van delta's de snelheid van de motoren aanpast. De robot komt hierdoor altijd wel goed uit, ook al is het misschien wat langzamer. De code die er voor zorgt dat de robot naar de bal rijdt is te zien in figuur 16.

In de code is te zien dat de PID-controller er nog wel in zit. En optioneel gebruikt kan worden. Dit kan worden gedaan door de “detector” klasse aan te maken met een True argument.

```
@app.route('/', defaults={'path': ''})
@app.route('/<path:path>')
def serve(path):
    if not path:
        path = "index.html"

    if path_to_file_exists(path):
        return send_from_directory(app.static_folder, path)
    else:
        return abort(404)

def path_to_file_exists(path):
    if not path:
        return False

    full_path = app.static_folder + "/" + path

    return os.path.exists(full_path)
```

Figuur 15: Flask code voor het serveren van de frontend applicatie.

```

def callback(self, detections):
    detection = self.get_nearest_detection(detections)

    if self.collected(detection):
        self.done()
        return

    direction, distance = self.get_control(detection)
    left, right = self.get_speed(direction, distance)

    self.zumo.run('left', left)
    self.zumo.run('right', right)

@staticmethod
def get_nearest_detection(detections):
    detection = max(detections, key=lambda d: d.width)
    return detection

def collected(self, detection):
    return detection.width > self.CLOSE_POINT

def done(self):
    self.zumo.run('stop')
    self.pause()
    print("Collected ball")

def get_control(self, detection):
    if self.use_pid:
        direction_control =
            self.direction_pid(detection.position)
        distance_control =
            self.distance_pid(detection.width)
    else:
        direction_control = detection.position - 0.5
        distance_control = 1 - detection.width
    return direction_control, distance_control

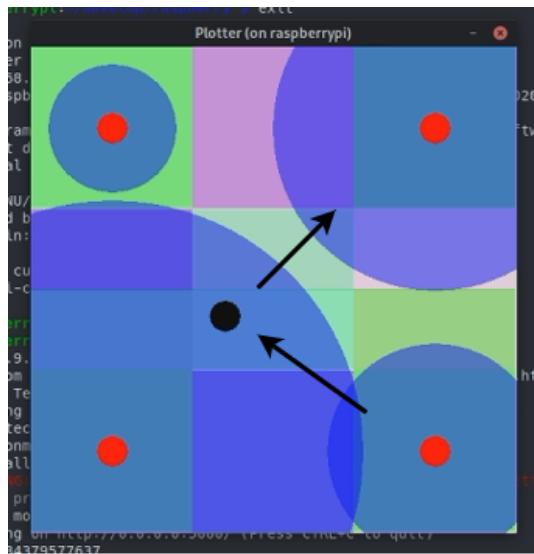
def get_speed(self, direction, distance):
    speed = abs(max(distance * self.MAX_SPEED, 150))
    left = abs(self.MIDDLE_POINT - direction) * speed
    right = abs(self.MIDDLE_POINT + direction) * speed
    return left, right

```

Figuur 16: De callback functie die acties onderneemd aan de hand van de detectie.

4.3 Ballen naar het verzamelpunt brengen

Om de ballen naar het verzamelpunt te brengen maakte we gebruik van meerdere services. Eentje daarvan was de positie service door middel van BLE. De redenen dat we deze service nodig hadden was zodat we een start positie konden definiëren, de auto een stukje laten rijden en daarna een eind positie berekenen.



Figuur 17: Pijlen met de berekende route

Nu we beide posities weten kunnen we gemakkelijk de hoek berekenen waar de robot naartoe rijdt. Dit kunnen we doen door eerst de hoek te berekenen van de eindpositie naar het verzamelpunt, en dan het verschil tussen beide hoeken te draaien. Daarna hoeven we alleen nog maar rechtdoor te rijden en te controleren of we dichtbij het verzamelpunt zijn door middel van de positie service.

Het enigste probleem waar we vaak tegen aanliepen was dat de kabels een grote invloed hadden in hoe recht de robot reedt. Hierdoor gebeurde het vaak dat de robot lichtelijk afdwaalt naar links of naar rechts en daardoor de berekening niet meer klopt.

4.4 Logica voor het verzamelpunt, zones en modus maken

Het kiezen van een verzamelpunt en zones kan worden gedaan door middel van de web app. De instellingen worden door de flask-server ingesteld en door de “Zones” klasse uitgelezen. Deze instellingen worden later gebruikt tijdens het berekenen van de route die de robot moet rijden naar een verzamelpunt of zone.

Nadat de robot een bal heeft verzameld moet de robot terug rijden naar de geselecteerde zone. Dit gebeurt door middel van de laatste bekende hoek van het terug brengen van de bal. Door deze hoek te combineren met de huidige positie en de positie van de zone kunnen we een nieuwe hoek berekenen waarnaar de robot naartoe moeten draaien. Vervolgens hoeven we alleen maar rechtdoor te rijden om bij onze eindbestemming uit te moeten komen.

4.5 Navigatie klasse maken

De navigatie klassen is een state-pattern die er voor zorgt dat al de hierboven genoemde technieken combineert in één systeem. De robot begint als eerste met het zoeken van een tennisbal door middel van object detectie. Vervolgens brengt de robot deze bal naar het geselecteerde verzamelpunt. Daarna reed de robot terug naar de geselecteerde zone. Vervolgens begon het hele traject weer opnieuw.



Figuur 18: De mogelijke state transitions

4.6 Binnen de zones blijven

De binnen de zones blijven is een beetje van scope veranderd. Dit kwam omdat het erg warrig gedefinieerd stond in het onderzoek. De uiteindelijke versie heeft voornamelijk de taak gekregen om naar de juiste zone te rijden en daar te gaan zoeken naar tennis ballen.

4.7 Obstakels ontwijken

Deze feature is er helaas niet in gekomen, dit komt voornamelijk omdat we geen tijd hadden om deze nice-to-have feature te implementeren. Verder namen we aan dat er niet echt obstakels op een tennisbaan zouden staan. Dus dit leek het ons niet een heel groot probleem om deze feature weg te laten.

5 Reflectie

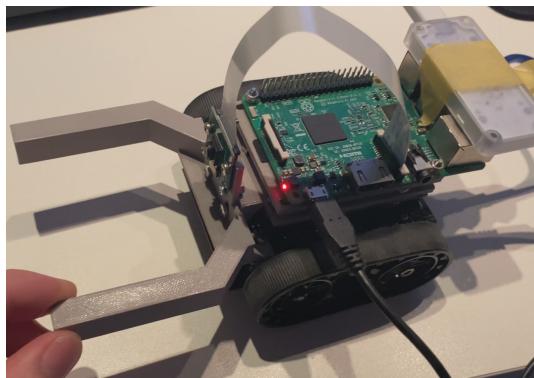
5.1 Wat ging goed

Bal navigatie

Het ophalen van de bal wilde we eerst met een PID-controller doen. Maar aangezien het tunen hiervan erg lang zou duren, hadden we maar besloten om het anders op te lossen. Tot onze verbazing werkte de nieuwe aanpak op basis van formules beter dan de PID-controller (los van het feit dat deze beter bijgesteld had kunnen worden).

De mount

De mount voor de Zumo werkte erg goed, op een enkele miscalculatie van de mounting holes aan de voorkant van de vork na (gelukkig konden we deze groter maken met een boor van 2mm). We zijn ook erg blij met hoe stevig de Raspberry Pi mount op het shield van de Arduino past.



Figuur 19: Mount gemonteerd op de Zumo robot

De webapplicatie

De web-applikatie was erg goed gelukt naar onze mening. De app zelf was makkelijk te begrijpen en kwam modern over. Dit komt mede door de single-page functionaliteit die met React was gemaakt. Het ontwikkelen van de app was ook niet erg lastig en duurde niet erg lang.

De integratie met de Python applicatie was ook erg makkelijk met de Flask backend-API. Wel vonden we het jammer dat Flask geen OOP oplossing biedt, aangezien de rest van de applicatie wel zo was geschreven.

5.2 Wat kon beter

Locatie

Door niet bij elkaar te zijn, werd de samenwerking op sommige fronten erg gecomprimeerd. Zowel testen als overleggen ging lastiger dan normaal en zorgde voor relatief minder efficiënte werkdagen.

BLE

Het probleem bij onze BLE oplossing was dat de signaal sterkte erg zwak was. Hierdoor ging het signaal vaak op in de ruis. Waardoor de accuratie van RSSI meting erg onbetrouwbaar werd op grotere afstanden.

Stappen motoren

Omdat er op de Zumo standaard DC motoren zitten hadden wij geen andere keuze dan deze te gebruiken. Helaas konden we deze niet vervangen met bijvoorbeeld stappenumotoren, hierdoor moesten wij voor sommige features van de robot meer ingewikkeldere/minder accurate oplossingen verzinnen. Een voorbeeld hiervan zou bijvoorbeeld zijn hoe vaak we de positie service gebruiken voor kleine veranderingen. In plaats van alleen de positie service gebruiken voor de initiële positie en verder rekenen op de stappen die de motor heeft gezet.

Batterijen

Tijdens het ontwikkelen van de robot konden we geen batterij gebaseerde stroom oplossing bouwen, dit kwam voornamelijk door dat niemand van ons de gereedschap thuis heeft liggen om zo'n opstelling te bouwen en te testen. Onze uiteindelijke oplossing hiervoor was om de Micro-USB kabel in de Raspberry Pi te laten zitten om als stroomtoevoer te dienen.

Helaas zorgde deze oplossing voor een ander groot probleem, namelijk dat de Zumo robot veel moeite had om de kabels waaraan die vast zat mee te bewegen, hierdoor werd het naar rijden erg beïnvloed.

Performance

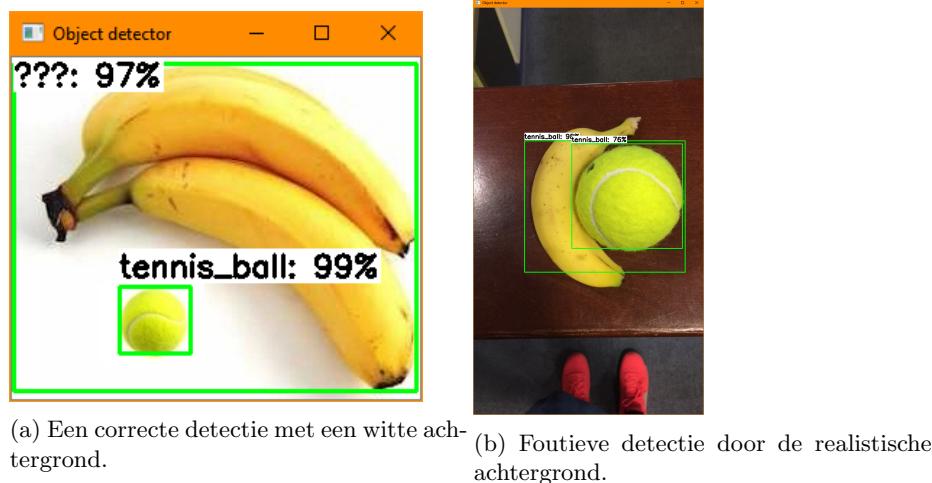
Tijdens het draaien van het programma merkte we dat de performance een stuk slechter was dan toen alles los van elkaar draaide. Dit komt waarschijnlijk door de hoeveelheid treads die we tegelijkertijd gebruiken. Bij de object detectie merkte we dat we in plaats van 15 FPS nog maar 5 FPS hadden. Dit hebben we uiteindelijk opgelost door de resolutie lager te zetten. Dit zorgt er helaas wel voor dat de detectie minder accuraat en minder ver zal werken.

Een manier om dit op te lossen was om de Raspberry Pi 4 te gebruiken. Naast meer ram heeft deze ook twee meer cores. Dit zou de performance van de applicatie goed kunnen bevorderen.

Machine learning

Het installatieproces van Tensorflow was erg omslachtig. Sergi had dit eerst op Linux geprobeerd, maar kwam er later pas achter dat de versie van Tensorflow uit maakte voor het trainproces. Aangezien de Tensorflow versie die we zochten (1.13) niet geïnstalleerd kan worden voor Python versie 3.8 moest er worden gedowngrade. Echter is dit erg lastig en niet gewenst op de versie van Linux die gebruikt werd. Er is toen voor gekozen om het trainen op Windows te doen, omdat de tutorial dit ook deed. Naast dat het installatieproces nog omslachtiger was is het uiteindelijk wel gelukt om het model te trainen.

Naast de installatie was het trainen ook niet helemaal optimaal. Het trainscript die we gebruikte kon als input alleen maar gelabelde selecties krijgen. Hierdoor konden we geen afbeeldingen zonder labels meegeven. Dit was handig geweest om hard-negatives te trainen zonder ze te labelen. Dit is wenselijk, omdat ze dan niet worden gedetecteerd en alleen onderdeel zijn van het trainingsproces. Nu moeten we de hard-negatives als selectie meegeven, wat er voor zorgt dat ze gedetecteerd zullen worden. Aangezien de hard-negatives erg verschillend waren wordt het detecteren van hard-negatives ook niet altijd goed gedaan. Ook hadden de hard-negatives een witte achtergrond, waardoor dit teveel meegenomen werd in het trainingsproces. In Figuur 20 is goed te zien hoe erg de achtergrond werd meegenomen in de detectie. Het is duidelijk dat de object detectie nu beter werkt op een achtergrond, wat niet realistisch is. In de toekomst zouden er afbeeldingen moeten worden gebruikt met een achtergrond, of zouden ze zelfs op tennisvelden moeten worden geplaatst.



Figuur 20: Het verschil tussen detectie met een witte- en een realistische achtergrond.

A Code & Middelen

Het volledige project is ontwikkeld via het Git platform GitHub. Dit deden we, omdat we er allebei bekend mee waren en omdat de taken goed verdeeld kunnen worden.

Het project is te vinden op: <https://github.com/philiipsens/TennisBallBot>

B Ontwerp

De web-applicatie was voorafgaand aan de ontwikkeling ontworpen in Figma. Hier is het design te zien: <https://www.figma.com/proto/mFk9zdLfH7LoPvKF8IsM8b/TennisBallBot-WebApp>

C Showcase

Tijdens het project moesten we de andere groepen een showcase van onze prototype laten zien. Deze is te zien op YouTube via deze link: <https://www.youtube.com/watch?v=x2dU6p4pzKc>

D Onderzoek

Start vanaf de volgende pagina

IROB Onderzoek

Joey de Ruiter & Sergi Philipsen

7 juli 2020



Wat is de beste manier om een robot efficiënt en snel tennisballen op te laten rapen?

Inhoudsopgave

1 Inleiding	3
2 Probleemanalyse	4
2.1 Aanleiding en context	4
2.2 Probleemstelling	4
2.3 Doelstelling	4
3 Theoretisch kader	5
3.1 Zumo-robot	5
3.2 Machine-learning	5
4 Onderzoekopzet en uitvoering	6
5 Resultaten	7
6 Onderzoek & Discussie	11
7 Conclusie	15

1 Inleiding

In dit onderzoek worden de mogelijkheden over het design en technische eigenschappen van de tennisballen opraaprobot aan de proef gesteld. Uit dit onderzoek volgt hoe het projectteam de best mogelijke technieken kiest voor het bouwen van een tennis bal opraap robot.

Er wordt als eerste gekeken naar het theoretisch kader, dat een wetenschappelijke basis vormt voor dit project. Hierin worden worden gebruikte begrippen gedefinieerd om zo duidelijk mogelijk een beeld te schappen over de verschillende termen die uiteindelijk gebruikt zullen worden als mogelijk referentie materiaal gebruikt kan worden tijdens het ontwikkelen van de robot.

Met de hoofdvraag en deelvragen die opgesteld zijn zal uiteindelijk een antwoord worden gevonden op de centrale vraag: *“Wat is de beste manier om een robot efficiënt en snel tennisballen op laten rapen?”*

Het uiteindelijke antwoord zal worden opgenomen in de conclusie en hierop zal een aanbeveling worden gedaan.

2 Probleemanalyse

2.1 Aanleiding en context

Voor de minor *Robotica, domotics en industriële automatisering* gaan we een robot maken die tennisballen op kan rapen. Om zeker te zijn hoe we dit product voor ons zien en aan gaan pakken, doen we dit onderzoek.

2.2 Probleemstelling

Het oprapen van tennisballen is een tijdsintensieve klus die de tennisspeler weerhoudt om door te gaan met sporten.

2.3 Doelstelling

De doelstelling is om onderzoek naar de beste mogelijkheden die wij kunnen toepassen om de robot te realiseren.

3 Theoretisch kader

3.1 Zumo-robot

De Zumo-robot of Zumobot is een robot gemaakt door Pololu (Pololu, g.d.). De robot bestaat uit een klein chassis gekoppeld met rupsbanden aan beide kanten en een kleine schans aan de voorkant. Tevens is de robot klein genoeg om mee te doen met mini sumo competities. Deze robot is goed geschikt als basis voor je eigen robot ontwerp.



3.2 Machine-learning

Machine learning is de studie van computeralgoritmen die automatisch verbeteren door ervaring. (Mitchell, 1997) Het wordt gezien als een subset van kunstmatige intelligentie. Machine learning-algoritmen bouwen een wiskundig model op basis van voorbeeldgegevens, ook wel 'training data' genoemd, om voorspellingen of beslissingen te nemen zonder hiervoor expliciet te zijn geprogrammeerd. (Koza, 1996)

4 Onderzoekspacet en uitvoering

Hoofdvraag

Wat is de beste manier om een robot efficiënt en snel tennisballen op te laten rapen?

De hoofdvraag zal worden beantwoord middels de antwoorden op de deelvragen en wordt vastgesteld in de conclusie.

Deelvragen

1. Wanneer moeten de tennisballen opgeraapt worden?
2. Waar op het tennisveld moeten de tennisballen worden opgeraapt?
3. Wat is de beste verzamelmethode die de Zumobot kan gebruiken zonder belast te raken?
4. Hoe kan de gebruiker op een eenvoudige wijze de robot bedienen?
5. Welke software en protocollen kunnen het best gebruikt worden voor de bediening van de robot?
6. Hoe weet de robot waar die zich bevindt op het tennisveld en wat is hier voor nodig?
7. Hoe worden de tennisballen gedetecteerd?
8. Welke machine-learning kan het best gebruikt worden voor de tennisballdetectie?
9. Hoe worden de detecties omgezet in acties?

De deelvragen zullen worden beantwoord door te discussiëren over de resultaten die zijn gevonden.

5 Resultaten

1. Wanneer moeten de tennisballen opgeraapt worden?

Uit eigen bevindingen is het mogelijk om verschillende manieren te gebruiken om te detecteren wanneer de tennisbal opgeraapt moet worden.

Het zou mogelijk zijn om de machine alle ballen aan het einde van een tennis oefening op te laten ruimen, maar het zou ook een mogelijkheid zijn om de bal tussendoor op te halen als er een fluitje af gaat, een druk op de knop van een mobiele app of afstandsbediening.

Ook zou de robot door de wedstrijd door de ballen op kunnen ruimen, als die niet in de weg zit.

2. Waar op het tennisveld moeten de tennisballen worden opgeraapt?

In de oefen en wedstrijd scenario's zijn op de volgende plekken tennisballen te vinden.

- Voor en achter het net
- Achter de zijlijn
- Achter de achterlijn
- Op het speelveld

Echter zal in beide gevallen de maten waarin dit is verschillen. Hierdoor moet er misschien worden gehouden aan hoe vaak de robot langs deze plekken gaat.

3. Wat is de beste verzamelmethode die de Zumobot kan gebruiken zonder belast te raken?

Om te kijken wat de beste verzamelmethode is moeten we eerst kijken naar hoe we de ballen willen opslaan. Hierbij komen de volgende ideeën naar boven:

- De ballen bewaren in een hoekje van het veld zodat ze later gemakkelijk opgeraapt kunnen worden.
- De ballen bewaren in een interne opslag van de robot zelf. Zodat die deze gelijk naar de spelers terug gebracht kan worden.
- De ballen in een vat stoppen zodat deze gemakkelijk opgeruimd kunnen worden.

Nu kunnen we het gaan houden over de mogelijke verzamelmethodes voor de tennisballen.

- De robot heeft een u-vormige ram aan de voorkant gemonteerd waarmee die de ballen over het veld kan duwen.
- De robot heeft een bus waarin die meerdere tennisballen mee kan nemen en ergens anders af kan leveren.
- De robot heeft een net waarin die meerdere tennisballen mee kan slepen en bij zich kan houden.
- De robot heeft een springveer op zich waarmee hij een tennisbal over het speelveld naar de juiste hoek kan schieten.

4. Hoe kan de gebruiker op een eenvoudige wijze de robot bedienen?

Met een afstandsbediening kan worden aangegeven welke “zone” van het veld opgeruimd moet worden. Tevens kan er ook met een app of een website worden bepaald waar opgeruimd moet worden, of kan de robot worden bestuurd.

5. Welke software en protocollen kunnen het best gebruikt worden voor de bediening van de robot?

Voor een afstandsbediening kan het beste via infra rood of 433Mhz de commando's worden verstuurd. Dit kan op de Arduino zonder dat er extra dingen nodig zijn, naast de sensoren zelf. Het zou ook mogelijk zijn om een website te gebruiken maar dan moet er gebruikt gemaakt worden van een Wi-Fi enabled chip, zoals een ESP8266 of Raspberry Pi.

6. Hoe weet de robot waar die zich bevindt op het tennisveld en wat is hier voor nodig?

De robot kan zichzelf kalibreren door eerst langs de randen van het veld te rijden. Eventueel kan de robot ook gebruik maken van de lichtbalk sensoren die onderaan de robot is gemonteerd. Op deze manier heeft de robot begrip voor waar de “zones” zich bevinden.

Met een hoog gemonteerde camera kan het volledige veld en de robot worden gedetecteerd. Dan is ten alle tijden duidelijk waar de robot zich bevindt.

Wellicht is het ook mogelijk om te berekenen in welk veld de Zumo robot zich bevindt doormiddel van een Raspberry Pi camera en wat ruimte bepaling.

Een alternatief zou zijn om een afstandssensor op de voorkant van de robot plaatsen, hiermee geef je hem de mogelijkheid om muren en andere obstakels en muren te ontwijken maar verliezen we een manier om accuraat bij te houden in welke zone we zitten.

Het is mogelijk om met BLE (Bluetooth Low Energy) beacons een verbinding

te maken met de Raspberry Pi. Door gebruik te maken van signaalsterkte en driehoeksmetingen kan de Raspberry Pi zichzelf localiseren (Leverege, 2019). Er zijn ook Bluetooth 5.1 oplossingen die kunnen meten wat de afstand en hoek van een verbinding is. Dit zou heel erg handig zijn aangezien er dan maar één beacon hoeft te worden geplaatst.

Een andere optie (die gesuggereerd werd door Vincent) is dat we een combinatie van gps of BLE en object detectie gebruiken om zo tot een relatief accuraat resultaat te komen. Hier zouden de palen van het net kunnen worden gebruikt als herkenningspunt.

7. Hoe worden de tennisballen gedetecteerd?

De tennisballen kunnen gevonden worden met de camera die gemonteerd wordt op de robot. Met de output van deze camera is het mogelijk om object detectie uit te voeren met behulp van machine learning. Hieruit hopen we de tennisballen te kunnen te identificeren.

Ook kan er gebruik worden gemaakt van een afstandssensor in combinatie met een kleurensensor om te kijken of er groen-gele objecten op het pad van de robot zijn.

8. Welke machine-learning kan het best gebruikt worden voor de tennisballendetetectie?

De tennisballen kunnen snel worden gedetecteerd met behulp van een CNN (Ren'o, 2018). Met edge detectie kunnen de ronde vormen makkelijk herkend worden. Ook kunnen de ballen herkend worden aan de duidelijke geel-groene kleur. Dit kan heel goed met OpenCV (Heywood, 2017)

Er kan ook gebruik worden gemaakt van een object detection classifier. Deze heeft het voordeel dat andere objecten als het net, de speler en andere obstakels ook herkend kunnen worden. (Rosebrock, 2020)

9. Hoe worden de detecties omgezet in acties?

Het zou mogelijk zijn om de robot meerdere statussen te geven, bijvoorbeeld eentje die zoekt naar een bal, eentje die een bal aan het tracken is en eentje die de bal aan het wegbrengen is.

Bij deze statussen is het belangrijk dat de robot zich op 1 onderdeel kan focussen en dat er niet andere acties van andere statussen “lekkken” in de huidige status.

Het zou ook mogelijk zijn om continue de dichtsbijzijnde bal te zoeken en deze op te halen en weg brengen. Een ander idee zou zijn om met een punten systeem te werken, die rekening houdt met de afstand en de hoeveelheid ballen. Zo kan

de robot zo efficiënt mogelijk te werk gaan.

De communicatie die op de robot zal plaatsvinden tussen de micro-controllers zal via de fysieke draden lopen. Voor het protocol hebben we dan veel verschillende keuzes, de meest voor de handliggende keuze lijkt ons UART (TI, 2010), dit komt omdat het een simpel te implementeren protocol is die weinig extra hardware nodig heeft. Verder hebben wij volledige controle over het dataformaat en baud-rate, hierdoor zijn de min-punten van UART eigenlijk een non-issue. Ook al zeggen we dat UART de meest voor de handliggende methode is moeten we nog steeds in ons achterhoofd houden dat andere protocollen ook een goede fit zouden zijn.

Een ander mogelijk protocol zou I^2C (NXP, 2014) zijn. Dit komt voornamelijk omdat het een heel populair en simpel protocol is waardoor het op vele platformen al een libary heeft. Tevens bied I^2C nog wat fijnen features, zoals dat de master de volledige bus beheert, hierdoor hoeven de slaves geen instellingen te delen. Helaas gaat de performance van dit protocol wel erg naar beneden als er veel data over de lijn wordt gestuurd.

6 Onderzoek & Discussie

1. Wanneer moeten de tennisballen opgeraapt worden?

Aangezien grote wedstrijden vaak al ballenjongens hebben en niet het probleem hebben dat er steeds veel ballen op moeten worden geraapt, focussen wij ons op de trainingen en amateurs. Voor de veiligheid van de robot, laten we deze langs de zijlijn wachten totdat deze aangestuurd wordt om de ballen te verzamelen.

2. Waar op het tennisveld moeten de tennisballen worden opgeraapt?

Het is in meerdere gevallen nodig dat de ballen over het gehele veld verzameld zullen worden. Echter is het wel zo dat bij trainingen merendeels van de ballen zich aan de tegengestelde zeide van de speler bevinden. Het lijkt ons handig om te werken met “zones” en aan de robot door te kunnen geven welke “zone” die moet opruimen.

3. Wat is de beste verzamelmethode die de Zumobot kan gebruiken zonder belast te raken?

Door de kleine motoren gemonteerd op de Zumo robot, hebben wij het gevoel dat als we een interne opslag er op monteren we niet genoeg kracht hebben om vooruit te komen. Tevens hebben we hetzelfde gevoel als we met een bal een schans op gaan rijden om deze uiteindelijk in een vat/opslag te krijgen.

Hierdoor blijft er eigenlijk maar één mogelijke oplossing over, en dat is de ballen bewaren in een hoekje van het veld zodat het makkelijk is op te ruimen door de tennisspeler.

4. Hoe kan de gebruiker op een eenvoudige wijze de robot bedienen?

Omdat we denken dat het fijn is voor de gebruiker om feedback te krijgen, lijkt het ons het handigst om dit met een web applicatie te doen. Dit is ook technisch mogelijk zonder dat wij extra hardware hoeven te bestellen.

5. Welke software en protocollen kunnen het best gebruikt worden voor de bediening van de robot?

Om een web applicatie te bouwen moeten we de applicatie natuurlijk bereikbaar maken voor een webbrowser, hiervoor zijn enkele mogelijkheden de meest simpele en meest robuuste techniek zou zijn om de Raspberry Pi zijn eigen Wi-Fi hotspot te laten creëren. Hiermee is het mogelijk om een verbinding te leggen tussen de robot en een extern apparaat zoals een smartphone of een laptop.

Met deze verbinding is het mogelijk om HTTP verkeer op te zetten tussen de webserver zoals een Nginx server en een internetbrowser.

6. Hoe weet de robot waar die zich bevindt op het tennisveld en wat is hier voor nodig?

Het kabileren van de robot op het veld zal niet werken, dit komt omdat de motoren van de robot DC motoren zijn en iedere “stap” die de robot zet is een niet uniforme afstand. Hierdoor is het niet mogelijk om te vertrouwen op de kalibratie.

Een hoog gemonteerde camera zou hier de technisch correcte oplossing zijn, met de camera is het mogelijk om continue de positie van de robot te weten en in welke zone deze aan het werk is. Helaas moeten we nog een camera ter beschikking hebben en daarnaast is het minder gebruiksvriendelijk om een camera op te moeten hangen iedere keer als een persoon wilt trainen.

De zones kunnen worden bepaald met een afstandssensor, deze kan tot vier en een halve meter detecteren. Zo kunnen we ongeveer bijhouden of de robot niet het veld betreedt door niet te ver van de muur te gaan. Ook kan de robot op deze manier lang de zijlijn blijven, al zal het niet heel accuraat zijn. Er is een mogelijkheid dat het veld geen muur heeft, in dit geval zal de robot niet goed kunnen bepalen waar die is.

Het is ook mogelijk om het met een meer “lokale” techniek indoor positioning toe te passen, hiervoor hebben we twee mogelijkheden gevonden die goed kunnen worden toegepast op dit project: een BLE gebaseerde oplossing of een Bluetooth 5.1 oplossing met Angle of Arrival (AoA) en Angle of Departure (AoD) oplossing.

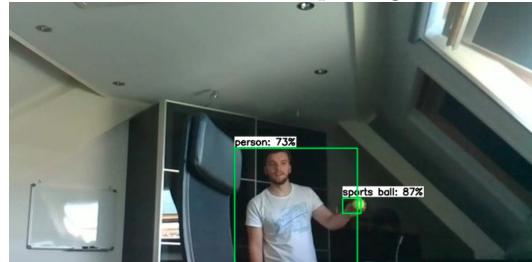
Het bluetooth 5.1 protocol is best nieuw, waardoor er nog weinig goedkope hardware voor beschikbaar is. Aangezien de oplossingen vaak zeldzaam en meer dan €40,- zijn, kunnen we deze manier van lokalisering maar beter achterwegen laten. Om BLE verder te onderzoeken hebben we een klein onderzoek gedaan met het opzetten van een implementatie van de BLE beacon techniek: iBeacon (Apple, 2013), met dit onderzoek kwamen wij er achter dat het mogelijk was om relatief accuraat de afstand tussen de Pi en de het apparaat (in ons geval een telefoon) te berekenen. Helaas werkt het minder als er een object in de weg staat van het signaal, maar we nemen aan dat er geen signaal blokkerende objecten op de tennisbaan staan.



7. Hoe worden de tennisballen gedetecteerd?

Voor de kleurensensor moet het object heel dichtbij komen. Dit is niet zo handig als er veel ballen in een keer moeten worden opgeruimd. Het lijkt ons daarom handiger om een Raspberry Pi met een camera te gebruiken. Het fijne hiervan is dat de Raspberry Pi ook de webserver bevat. Hierdoor kan heel veel logica op de Raspberry Pi worden gedaan, wat net wat fijner werkt. Zo hoeven we niet veel rekening te houden met opslag en kunnen we elke programmeertaal kiezen die we willen.

We hebben ook de getest of het mogelijk is om object detectie te doen op een relatief grote afstand. Het resultaat hiervan was dat een persoon en bal met een goed getrainde dataset (coco dataset) nog op een verre afstand gedetecteerd werd. Hierdoor nemen wij de aannamen dat als we een goed getrainde dataset op tennisballen hebben dat dit ook werkt op een grotere afstand.



8. Welke machine-learning kan het best gebruikt worden voor de tennisballendetetectie?

Om een accuraat en efficiënte robot te maken lijkt het ons handig om gebruik te maken van de snelste manier van machine learning. Er zijn twee state of the art modellen die gebruikt kunnen worden voor object detectie op kleine apparaten zoals de Raspberry pi. Dit zijn SSD en YOLO (Gunnarsson, 2019). Uit een vergelijking blijkt dat SDD toch net iets sneller en accurater is dan YOLO

op kleine apparaten. Ook willen wij gebruik maken van de Google Coral die de performance van de Raspberry Pi aanzienlijk kan verbeteren. Deze limiteert ons enigzins met modellen. Gelukkig is er wel een SSD model voor. (Tensorflow, 2020) Hierom hebben wij besloten om gebruik te maken van het volgende model: *ssd_mobilenet_v2_quantized_coco*.

9. Hoe worden de detecties omgezet in acties?

Het lijkt ons het beste om de robot meerdere statussen te geven, hierdoor kunnen we echt de focus leggen op de toestand die belangrijk is op het huidige moment. Zonder dat het een “spaghetti” wordt met wat de robot op dat moment moet doen.

Voor communicatie tussen de Arduino en de Raspberry Pi gaan we I^2C gebruiken, de meest voornaamste redenen hiervoor is dat dit de standaard is binnen Wire library van de Arduino en het ook gemakkelijk in te stellen is op de Raspberry Pi. Verder heeft het min-punt van een lage transfer rate bij veel data geen grote invloed, aangezien er maar af en toe kleine pakketten naar de Arduino wordt gestuurd met alleen maar aansturing commando's.

7 Conclusie

Om antwoord te geven op de hoofdvraag “*Wat is de beste manier om een robot efficiënt en snel tennisballen op te laten rapen?*” hebben wij geconcludeerd dat we voor het verzamelen de u-vormige ram het beste is. Dit komt voornamelijk omdat wij erg gelimiteerd zijn in het gebruik van materialen en hardware. Het verzamelen zal de robot in zones doen, we nemen namelijk aan dat de ballen ongeveer bij elkaar liggen, hierdoor zou de robot iets efficiënter zijn werk kunnen doen. De zone waar de robot zal opruimen, kan worden aangegeven door de gebruiker via een web applicatie. Om te bepalen in welke zone de robot zich bevindt gaan we gebruik maken van BLE lokalisering, ook al is dit niet super accuraat, we gaan er van uit dat dit accuraat genoeg is voor onze toepassing.

In de zone zal de robot gebruik maken van de Raspberry Pi en de camera om de ballen in beeld te brengen. Met object detectie bepalen we waar de ballen zijn en waar de robot naartoe zal rijden. De ballen zullen vervolgens naar een verzamelpunt worden gebracht.

Literatuurlijst

- Apple. (2013). *iBeacon*. Verkregen 29 mei 2020, van <https://developer.apple.com/ibeacon/>
- Gunnarsson, A. (2019). *Real time object detection on aRaspberry Pi*. Verkregen 27 mei 2020, van <https://www.diva-portal.org/smash/get/diva2:1361039/FULLTEXT01.pdf>
- Heywood, M. (2017, december 31). *Object Detection and Tracking with OpenCV and Python*. Verkregen 26 mei 2020, van <https://www.bluetin.io/opencv/object-detection-tracking-opencv-python/>
- Koza, J. R. e. (1996). *Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming*. https://doi.org/10.1007/978-94-009-0279-4_9
- Leverege. (2019, januari 4). *Indoor Positioning with Bluetooth Low Energy (BLE)*. Verkregen 28 mei 2020, van <https://www.iotforall.com/indoor-positioning-bluetooth-low-energy-ble/>
- Mitchell, T. (1997). *Machine Learning*. Verkregen 25 mei 2020, van <http://www.cs.cmu.edu/~tom/mlbook.html>
- NXP. (2014, april 4). *I2C-bus specification and user manual*. Verkregen 26 mei 2020, van <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>
- Polulu. (g.d.). *Zumo-bot*. Verkregen 25 mei 2020, van <https://www.pololu.com/category/129/zumo-robots-and-accessories>
- Ren' o, V. e. (2018). *Convolutional Neural Networks based ball detection in tennis games*. Verkregen 26 mei 2020, van http://openaccess.thecvf.com/content_cvpr_2018_workshops/papers/w34/Reno_Convolutional_Neural_Networks_CVPR_2018_paper.pdf
- Rosebrock, A. (2020, januari 27). *YOLO and Tiny-YOLO object detection on the Raspberry Pi and Movidius NCS*. Verkregen 26 mei 2020, van <https://www.pyimagesearch.com/2020/01/27/yolo-and-tiny-yolo-object-detection-on-the-raspberry-pi-and-movidius-ncs/>
- Tensorflow. (2020). *Tensorflow detection model zoo*. Verkregen 27 mei 2020, van https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md
- TI. (2010, november). *Universal Asynchronous Receiver/Transmitter*. Verkregen 26 mei 2020, van <https://www.ti.com/lit/ug/sprugp1/sprugp1.pdf?ts=1590482964077>

Referenties

- [1] *Arduino Uno*. URL: <https://store.arduino.cc/arduino-uno-rev3>.
- [2] *Beacontools*. URL: <https://pypi.org/project/beacontools/>.
- [3] *Coco-dataset*. URL: <https://cocodataset.org>.
- [4] *EdjeElectronics Tutorial*. URL: <https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi>.
- [5] *Flask*. URL: <https://flask.palletsprojects.com>.
- [6] *Google Coral*. URL: <https://www.coral.ai/>.
- [7] *iBeacon*. URL: <https://developer.apple.com/ibeacon/>.
- [8] *OpenCV*. URL: <https://opencv.org/>.
- [9] *Raspberry Pi*. URL: <https://www.raspberrypi.org/>.
- [10] *React*. URL: <https://reactjs.org/>.
- [11] *Tensorflow Lite*. URL: <https://www.tensorflow.org/lite/>.
- [12] *Zumo-bot*. URL: <https://www.pololu.com/category/129/zumo-robots-and-accessories>.