

PYTHON ROBOTICS

WORKSHOP 3

© 2022, 2023 Karel Dupain & Joep Suijs
Beeld- en geluidopnames niet toegestaan

HUISWERK 1

- Maak een statemachine voor een 'Duits' verkeerslicht
- Pas het verkeerslicht aan zodat je met knop A kunt schakelen naar 'oranje knipperen' en met knop B terug naar normaal gebruik.

Bonuspunten:

- De RGB leds hebben 8 kleur-combinaties. Maak een statemachine die deze om de beurt toont. Ga naar de volgende kleur als je op knop A drukt.

HUISWERK 2

- Maak een statemachine voor een 'Duits' verkeerslicht
- Pas het verkeerslicht aan zodat je met knop A kunt schakelen naar 'oranje knipperen' en met knop B terug naar normaal gebruik.

Bonuspunten:

- De RGB leds hebben 8 kleur-combinaties. Maak een statemachine die deze om de beurt toont. Ga naar de volgende kleur als je op knop A drukt.

HUISWERK 2 & KNOP B...

- Lezen van knop B op dezelfde manier als knop A werkt niet...
- Het ontbreken van een externe pull-up lijkt een oorzaak.
- Maar... met het inschakelen van de interne pull-up werkt het nog niet...
- Gelukkig heeft de microbit library een class voor de knoppen, met methodes om ze uit te lezen (niveau en one-shot).

Classes

`class Button`

Represents a button.

Note

This class is not actually available to the user, it is only used by the two button instances, which are provided already initialized.

`is_pressed()`

Returns `True` if the specified button `button` is currently being held down, and `False` otherwise.

`was_pressed()`

Returns `True` or `False` to indicate if the button was pressed (went from up to down) since the device started or the last time this method was called. Calling this method will clear the press state so that the button must be pressed again before this method will return `True` again.

HUISWERK 3

- Maak een statemachine voor een 'Duits' verkeerslicht
- Pas het verkeerslicht aan zodat je met knop A kunt schakelen naar 'oranje knipperen' en met knop B terug naar normaal gebruik.

Bonuspunten:

- De RGB leds hebben 8 kleur-combinaties. Maak een statemachine die deze om de beurt toont. Ga naar de volgende kleur als je op knop A drukt.

IMPORT

```
1  
2 from MyQueen import *  
3 Sm = StateMachine()  
4
```

```
8 # MyQueen.py  
9 class StateMachine:  
10  
11     # -----  
12     def init (self)
```


IMPORT

```
1  
2 from MyQueen import *  
3 Sm = StateMachine()  
4
```

```
4  
5 import MyQueen  
6 Sm = MyQueen.StateMachine()  
7
```

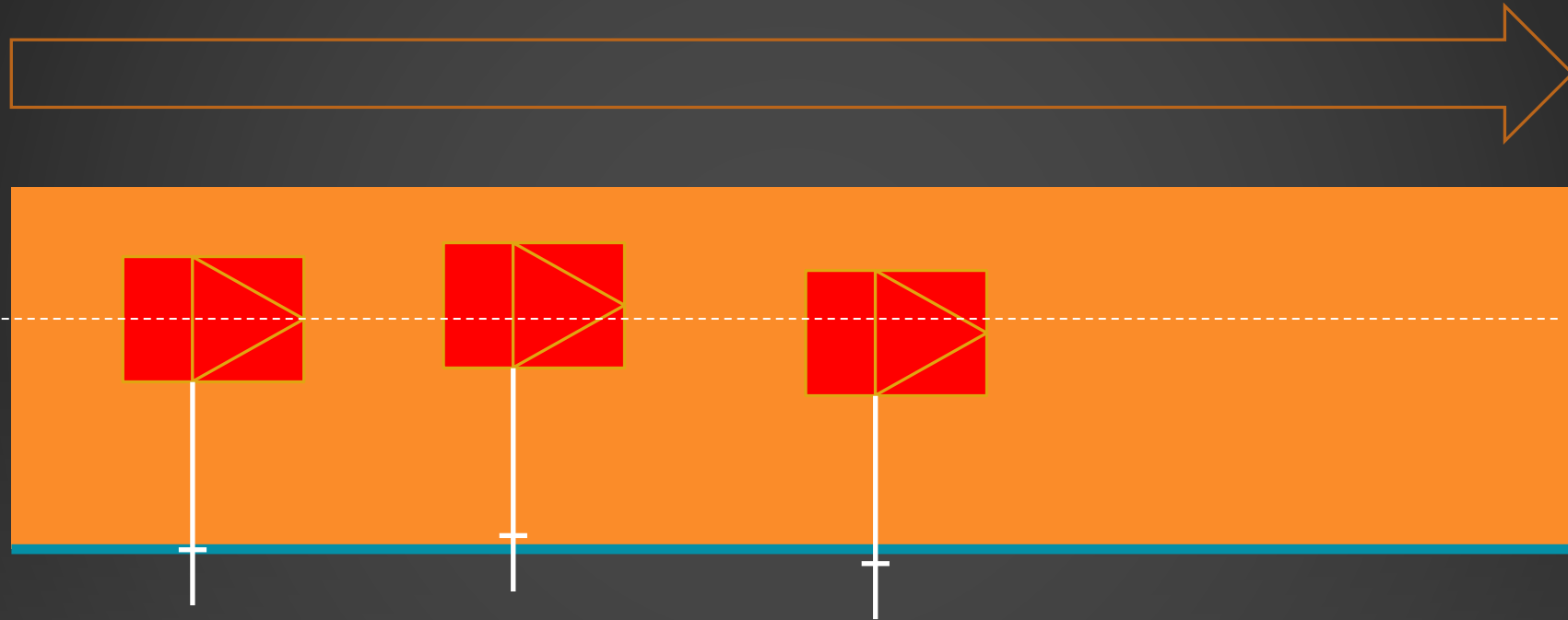
IMPORT

```
1  
2 from MyQueen import *  
3 Sm = StateMachine()  
4
```

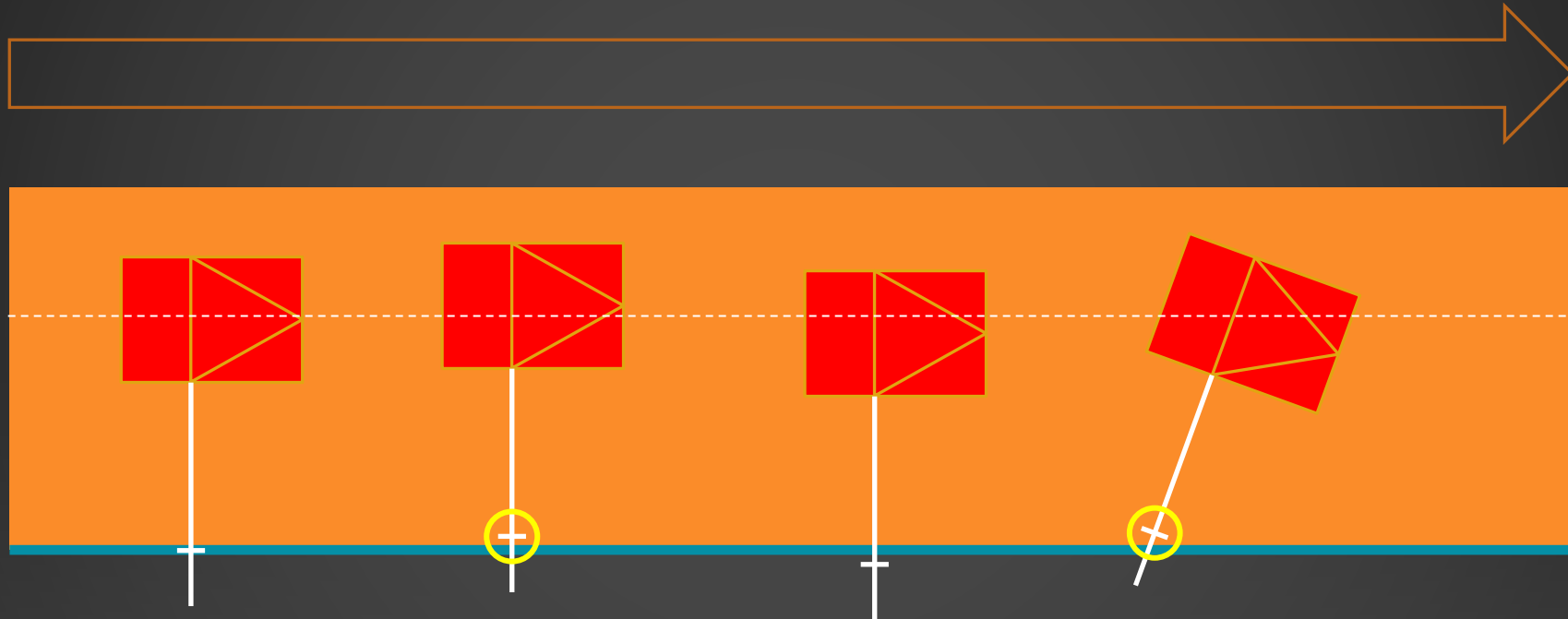
```
4  
5 import MyQueen  
6 Sm = MyQueen.StateMachine()  
7
```

```
7  
8 import MyQueen as MaqueenPlus  
9 Sm = MaqueenPlus.StateMachine()  
10
```

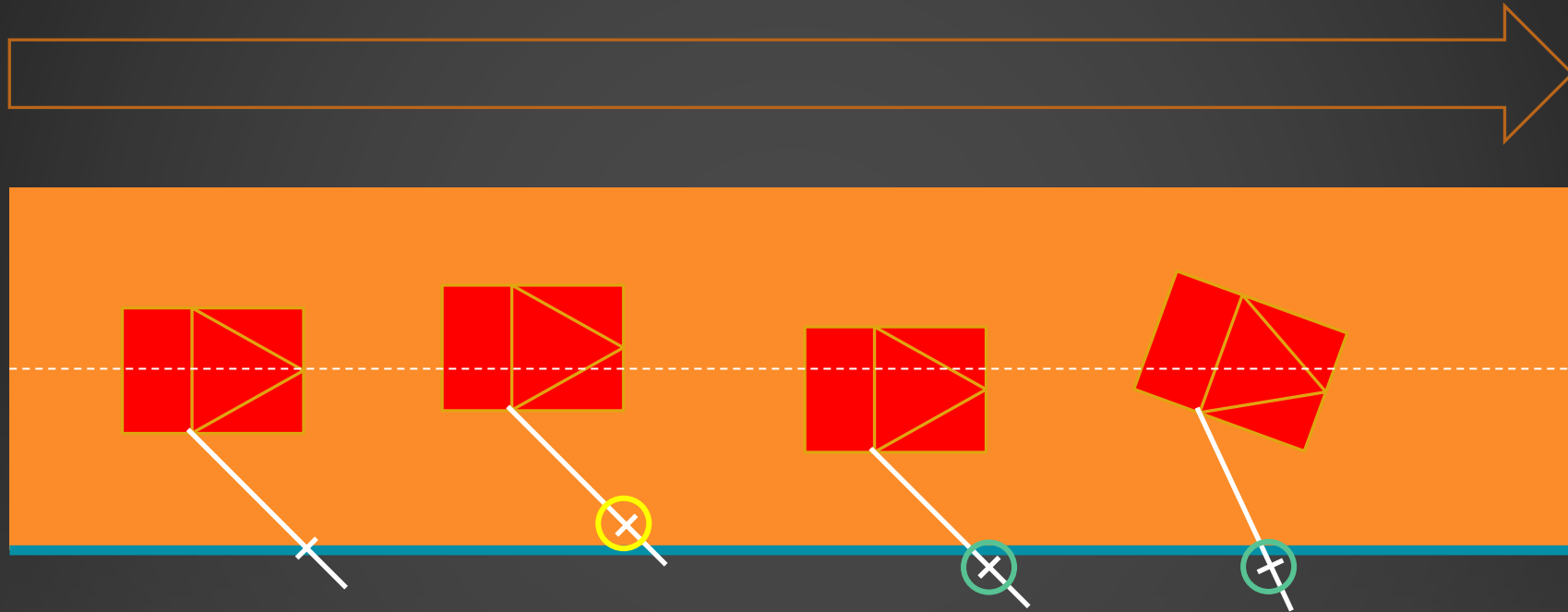

WANDVOLGEN (1)



WANDVOLGEN (2)

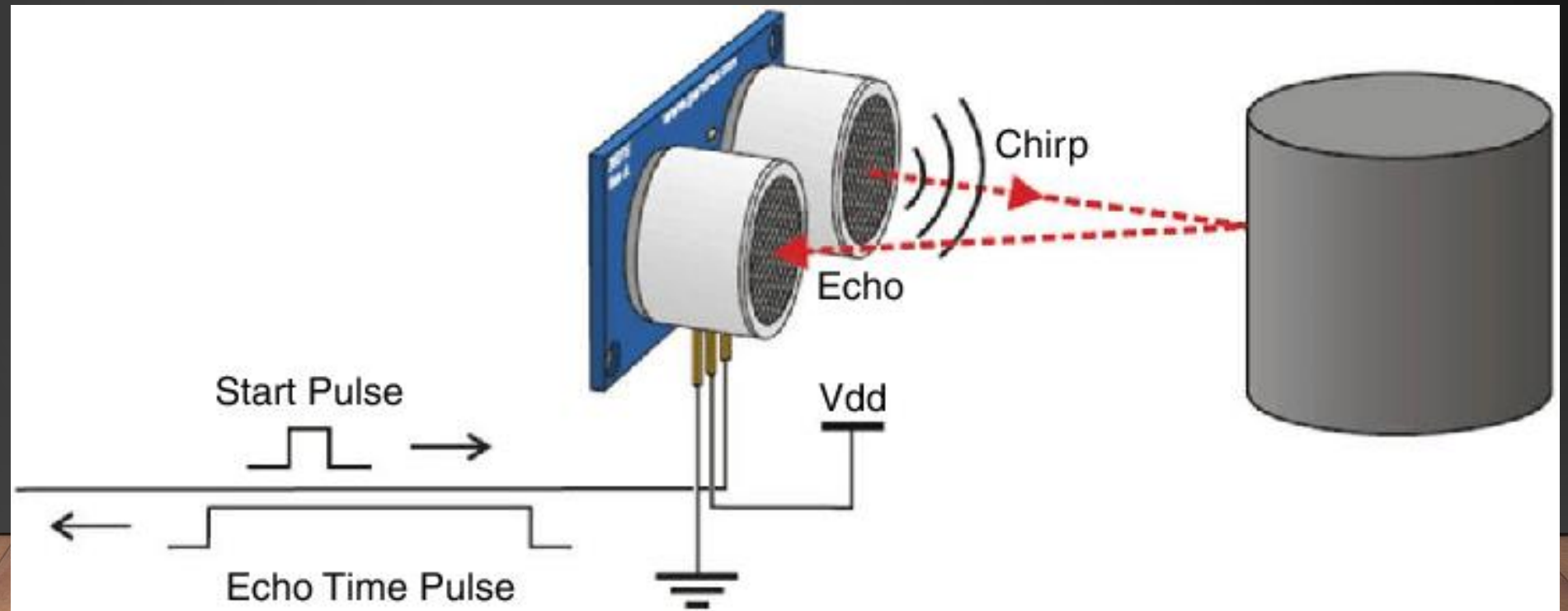


WANDVOLGEN (3)



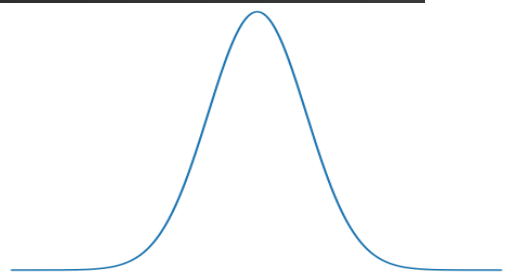
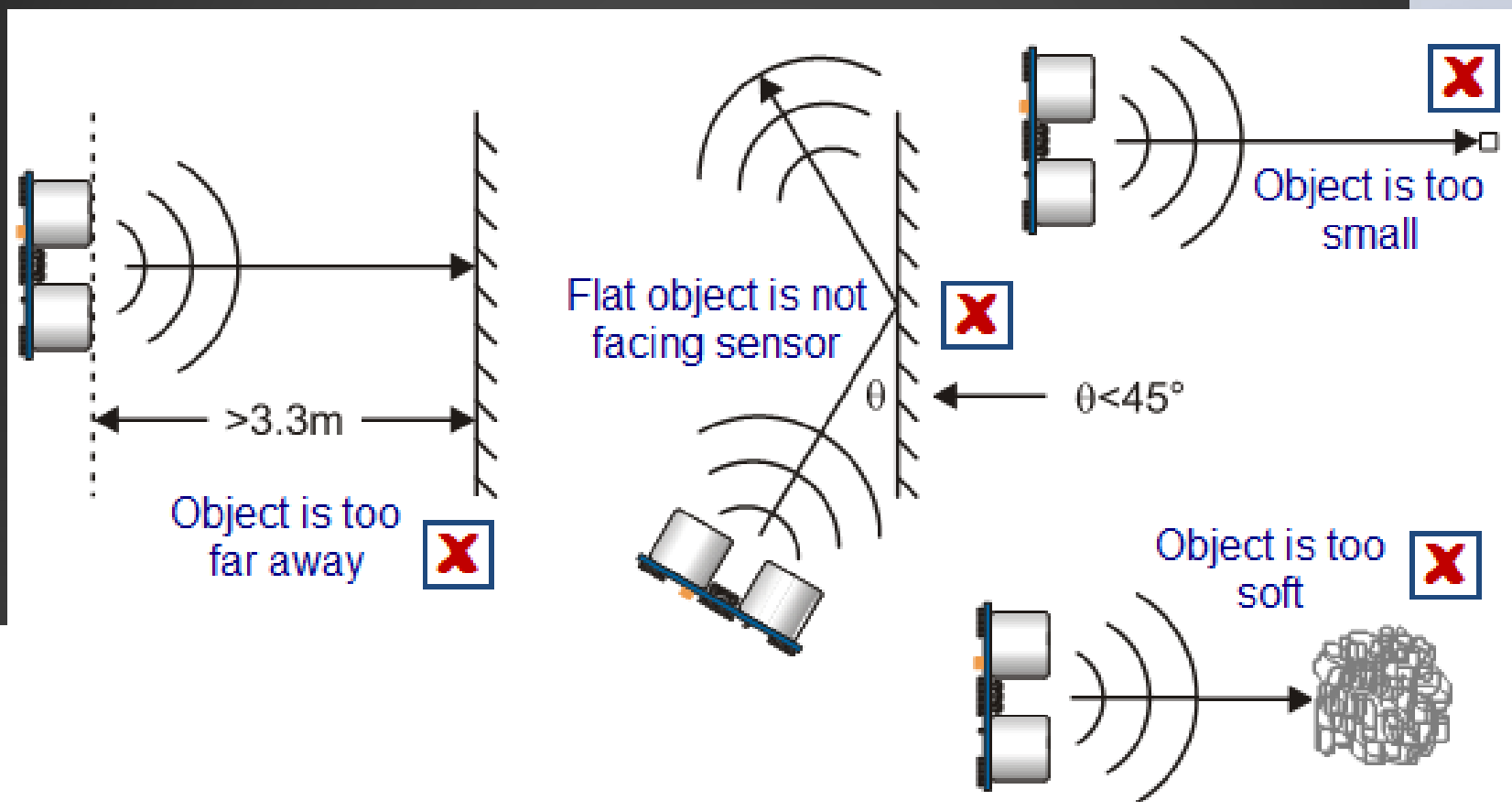
ULTRASOON AFSTAND METEN

- Meet looptijd van het verzenden van een ultrasoon burst tot ontvangst echo
- Geluidssnelheid 343 m/sec (@20 graden in droge lucht)
- Oftewel 343 mm / miliseconde



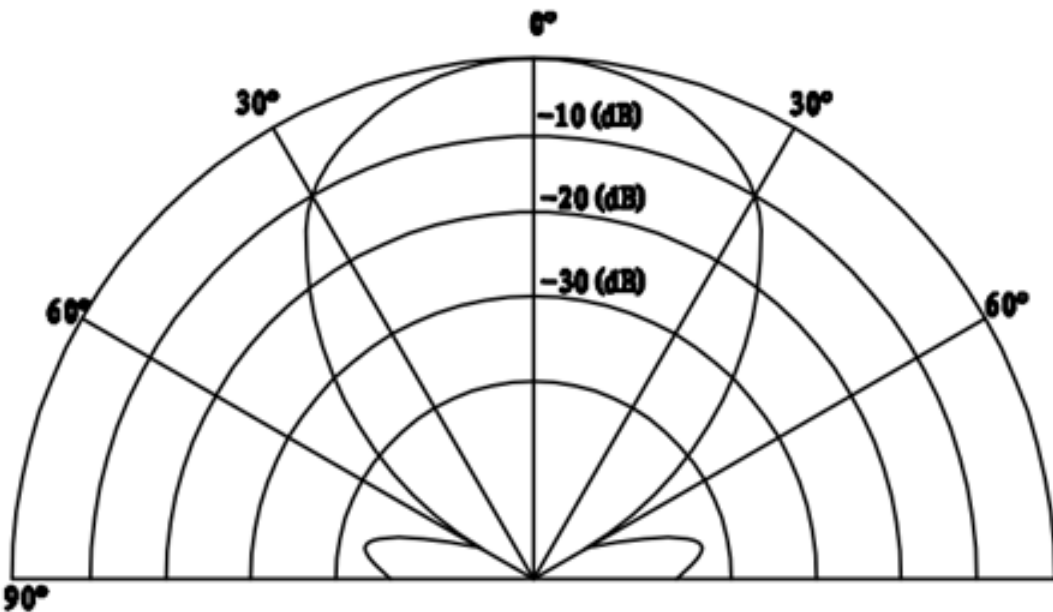
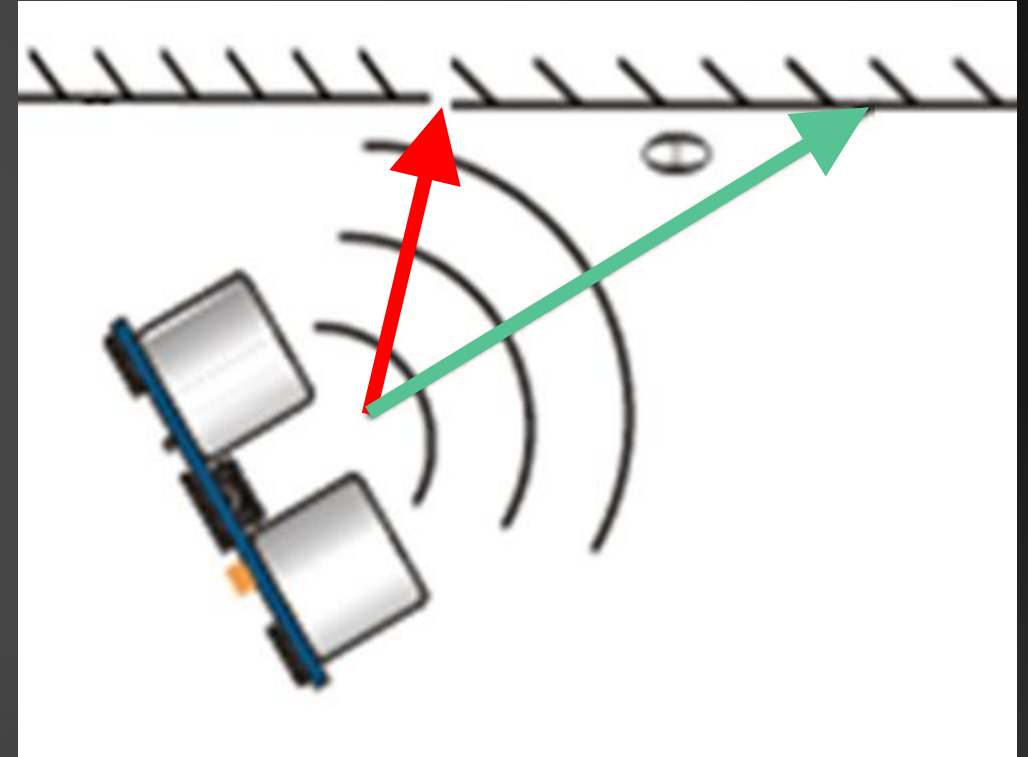


ULTRASOON



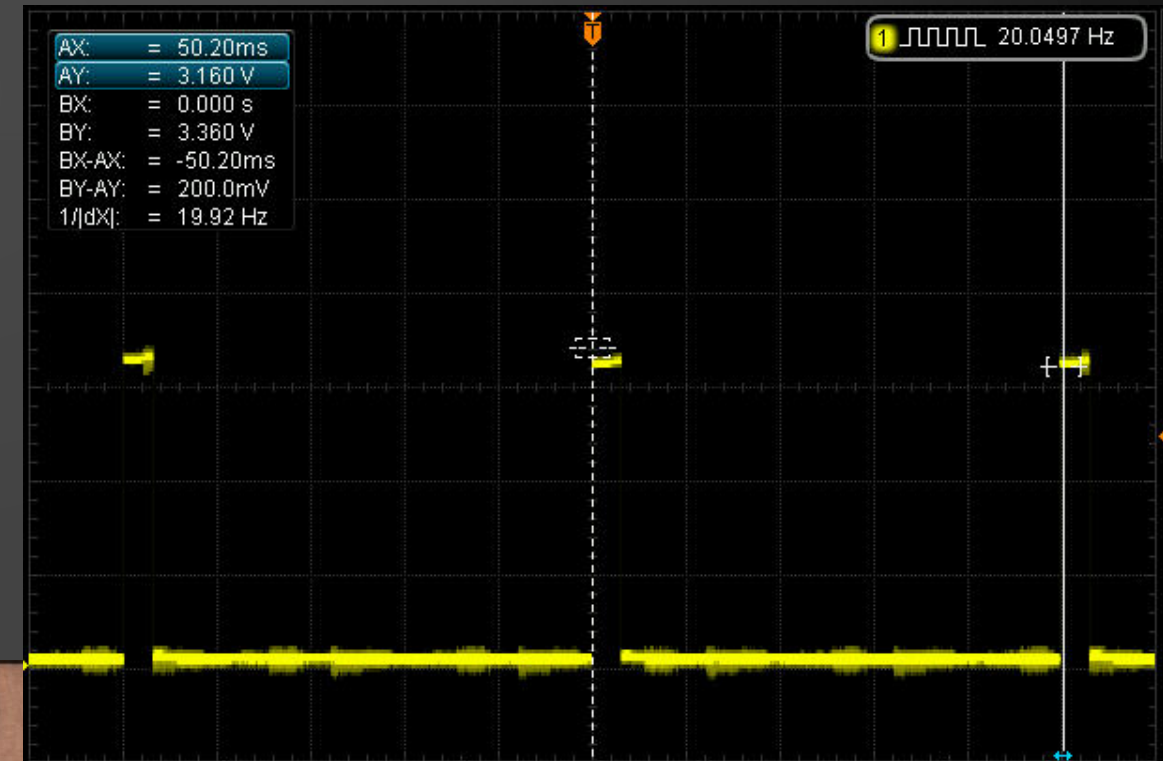
ULTRASOON WANDVOLGEN

- Brede openingshoek
- Zwakke reflectie onder 40 graden
- Sterke reflectie op onregelmatigheden

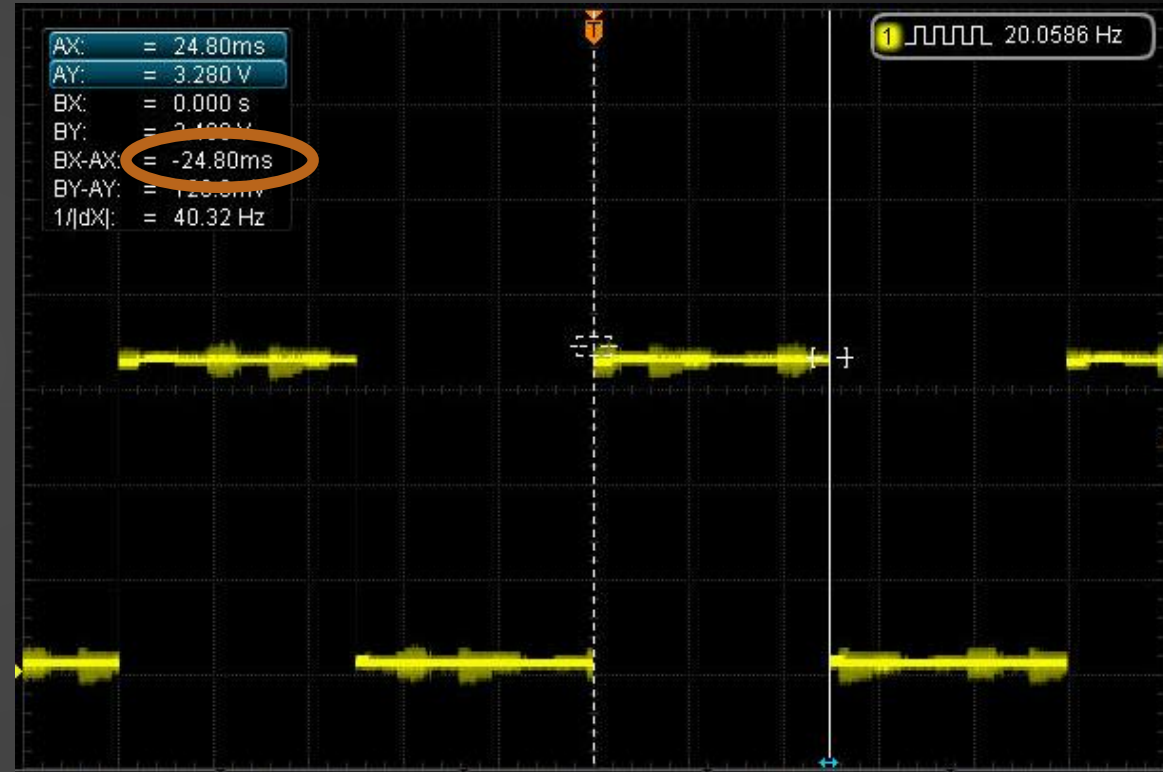
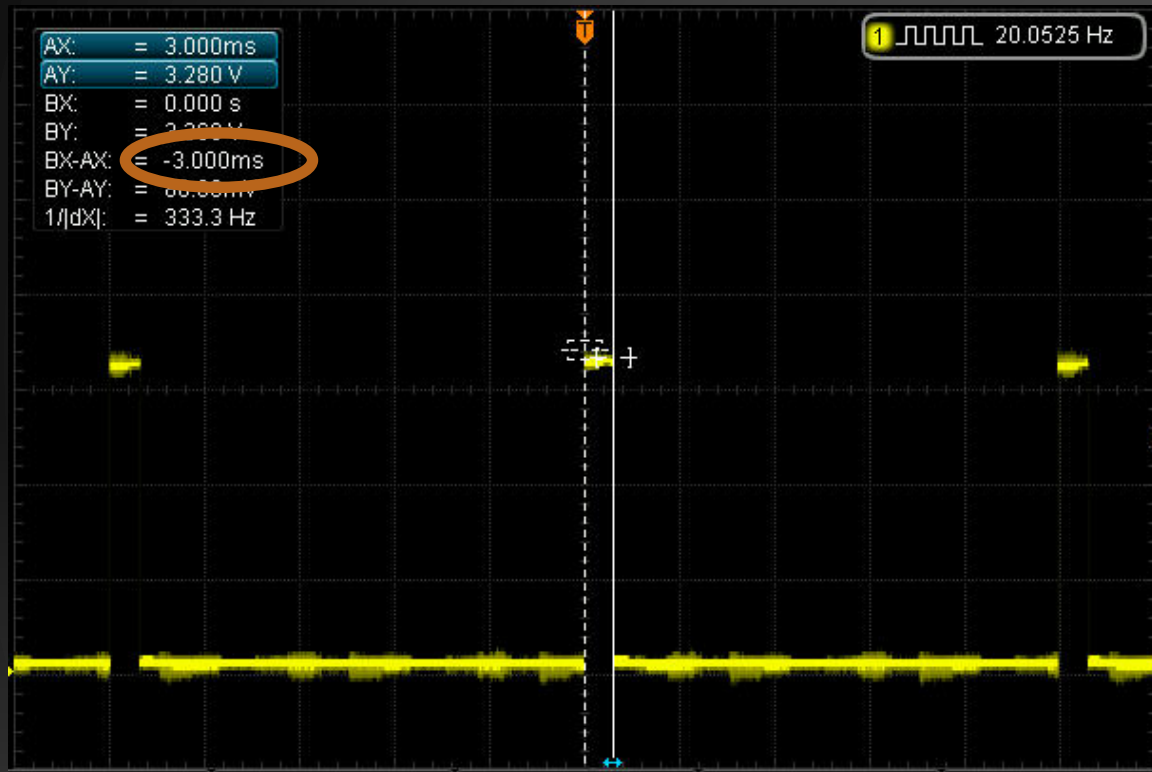


GY-53

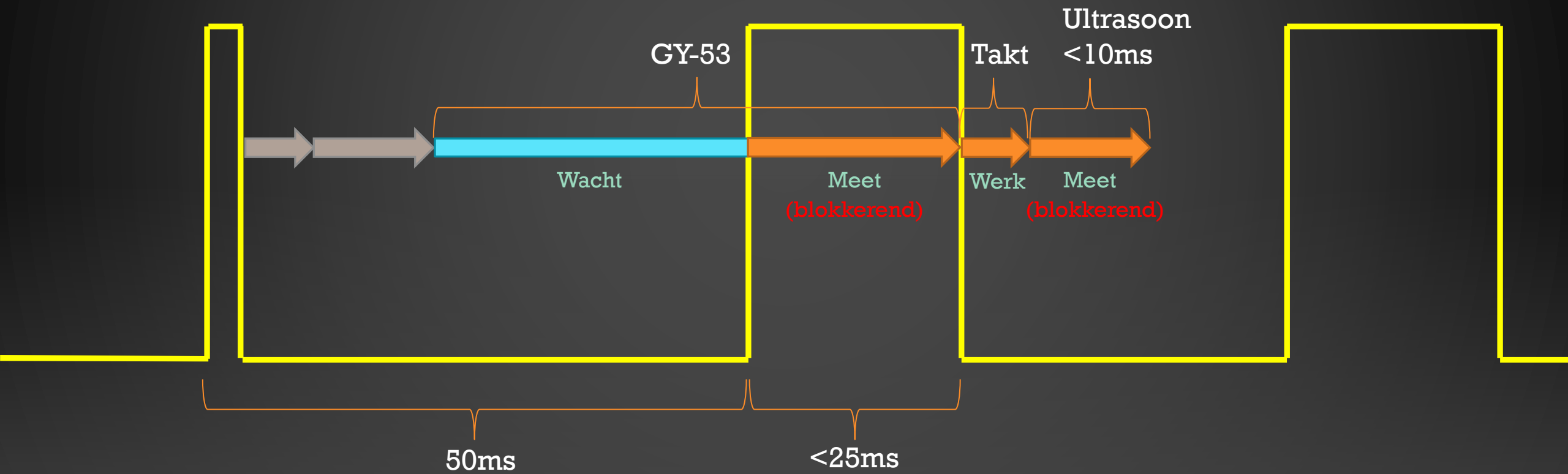
- Time of flight - VL53L0x-gebaseerd
- Eigen (pre)processor
- Eenvoudige interface



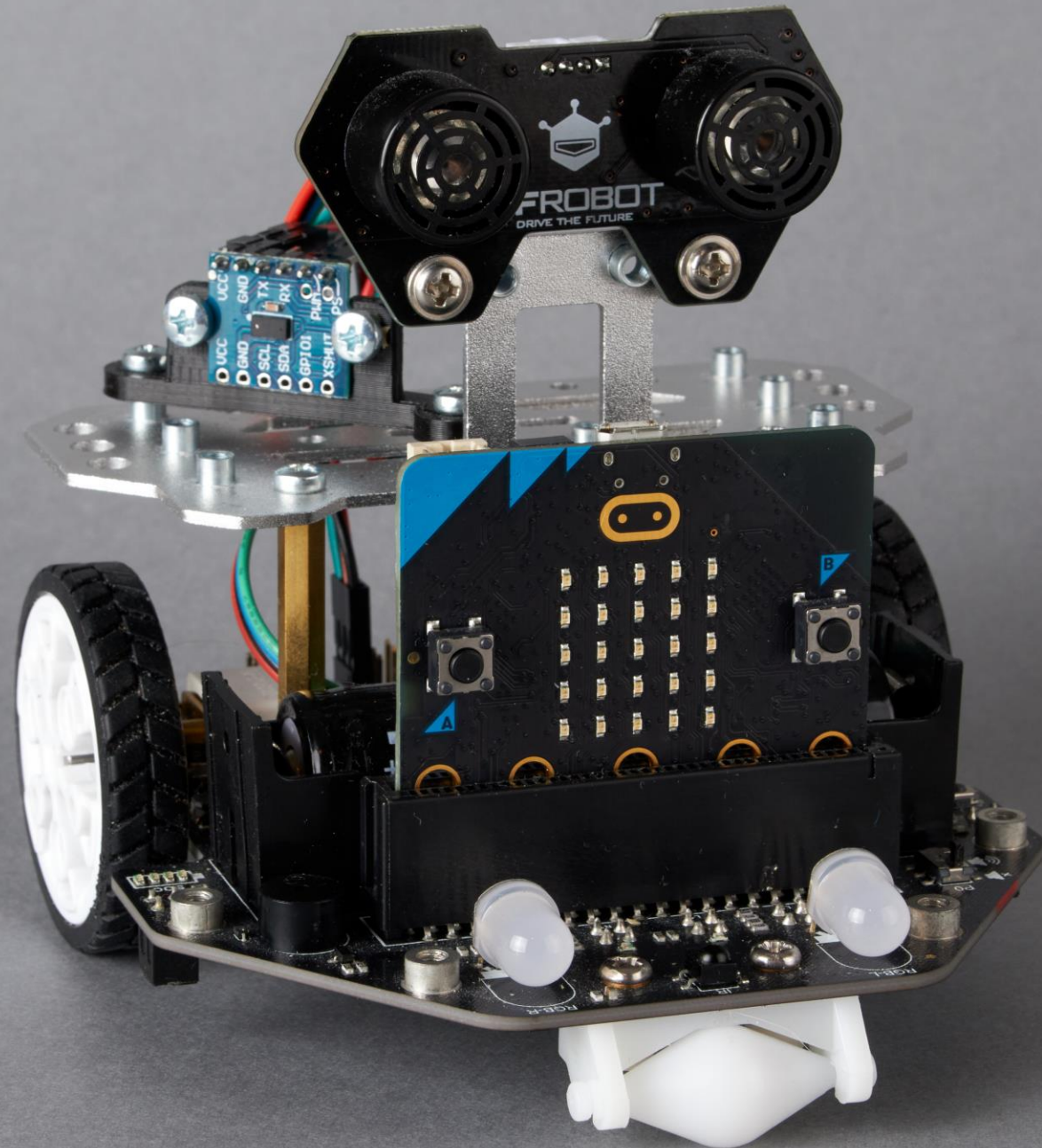
GY-53 PWM OUTPUT



TIMING



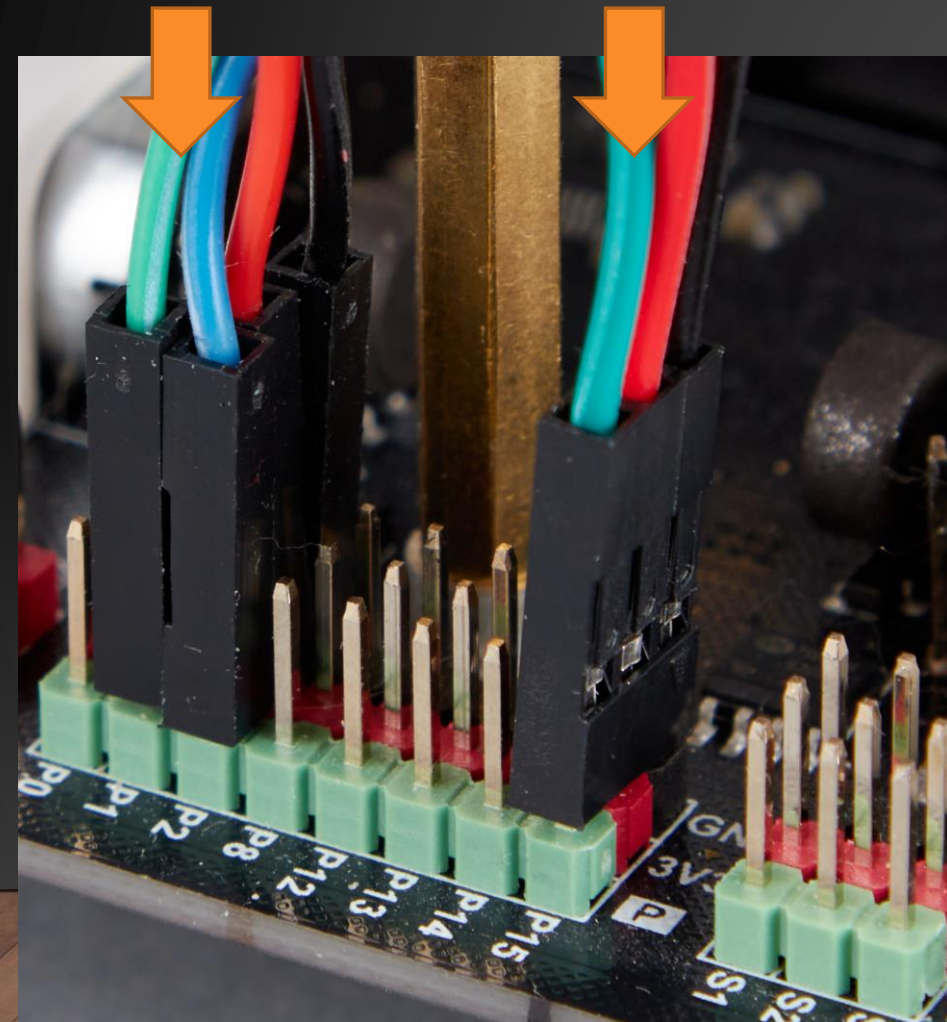
Dus in worst-case-scenario heb je 15 ms voor takt (werk, regellus).



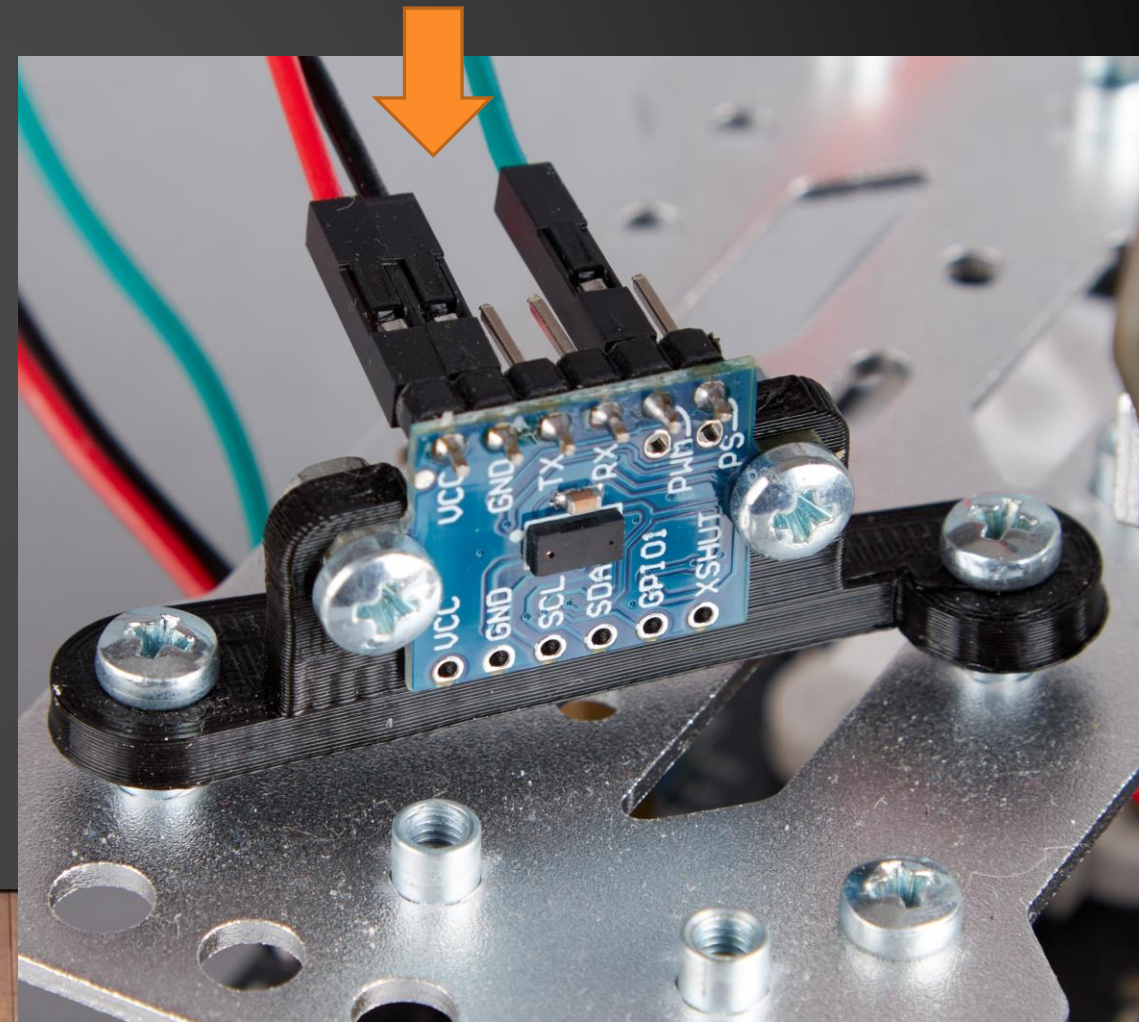
OEFENING – SENSOREN AANSLUITEN & TESTEN

Ultrason

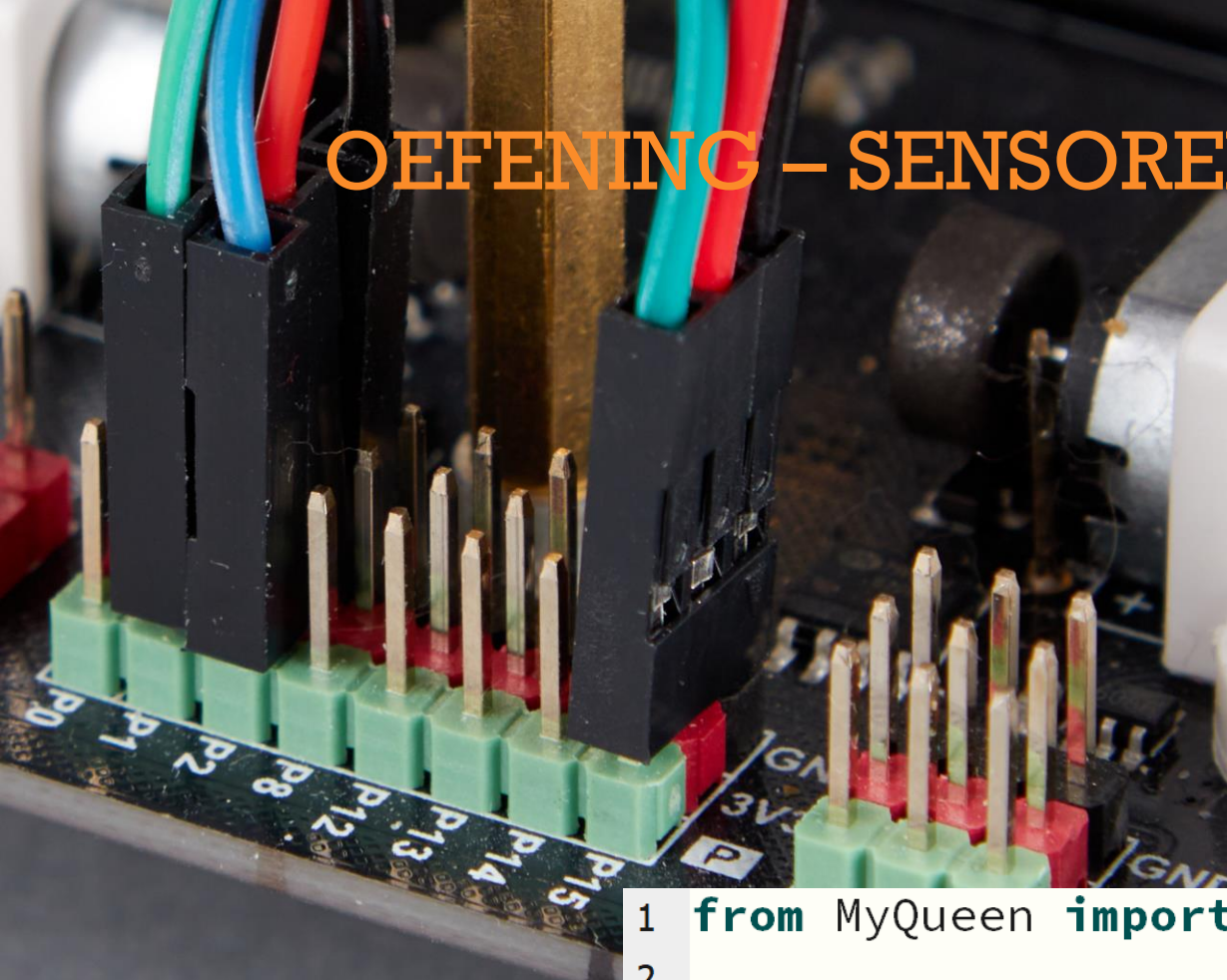
GY-53



GY-53



OEFENING – SENSOREN AANSLUITEN & TESTEN



```
1 from MyQueen import *
```

```
2
```

```
3 Sensoren = TSensors()
```

```
4 print("SensorTest")
```

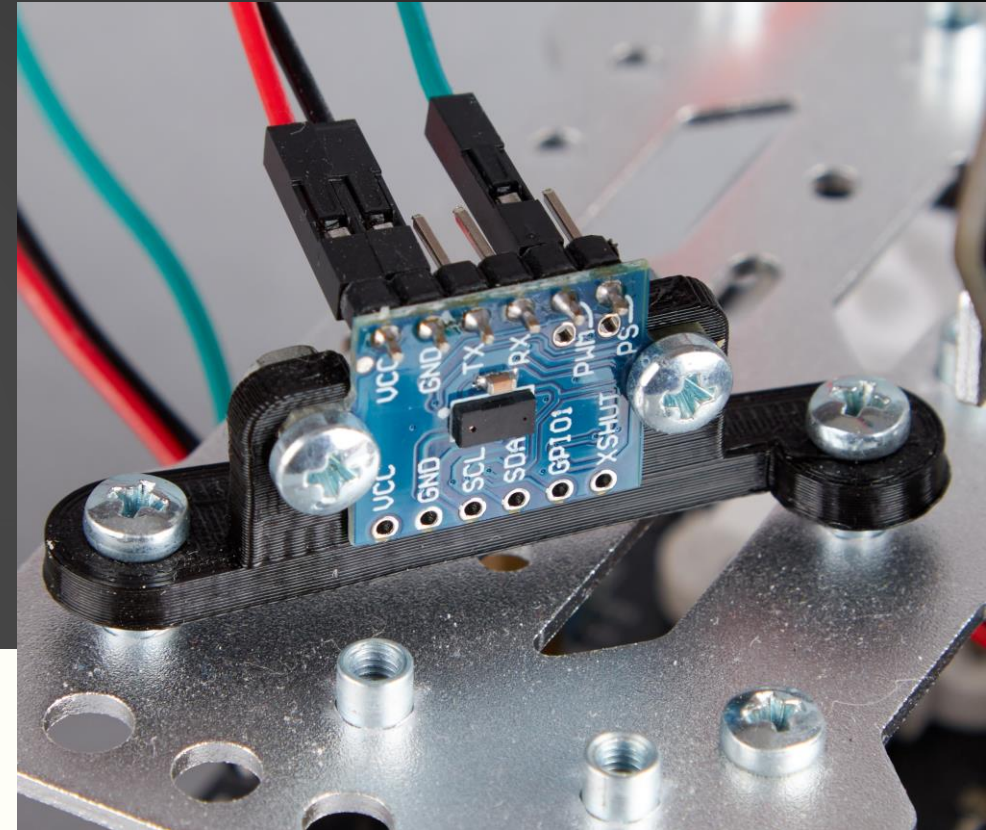
```
5
```

```
6 for i in range(1, 5) :
```

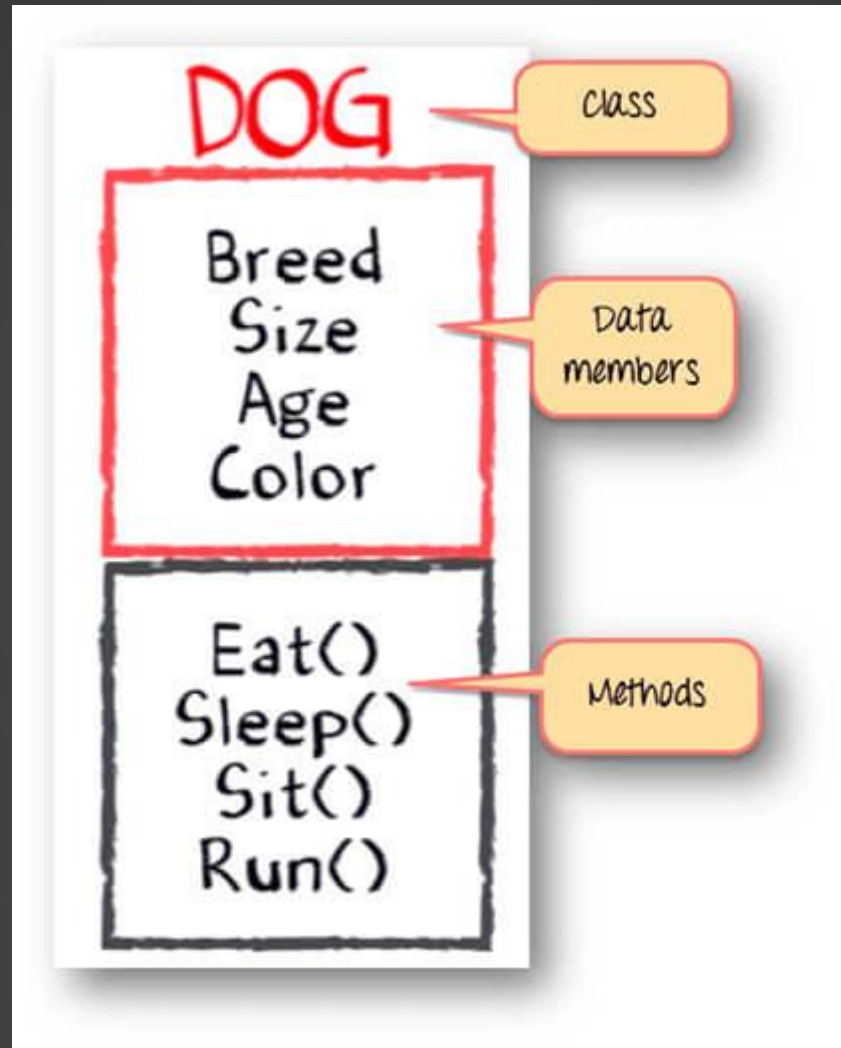
```
7     Sensoren.Takt()      # volgt tevens timing van GY-53 pwm signaal
```

```
8     print("Afstanden: ", Sensoren.GyDistance, Sensoren.UsDistance)
```

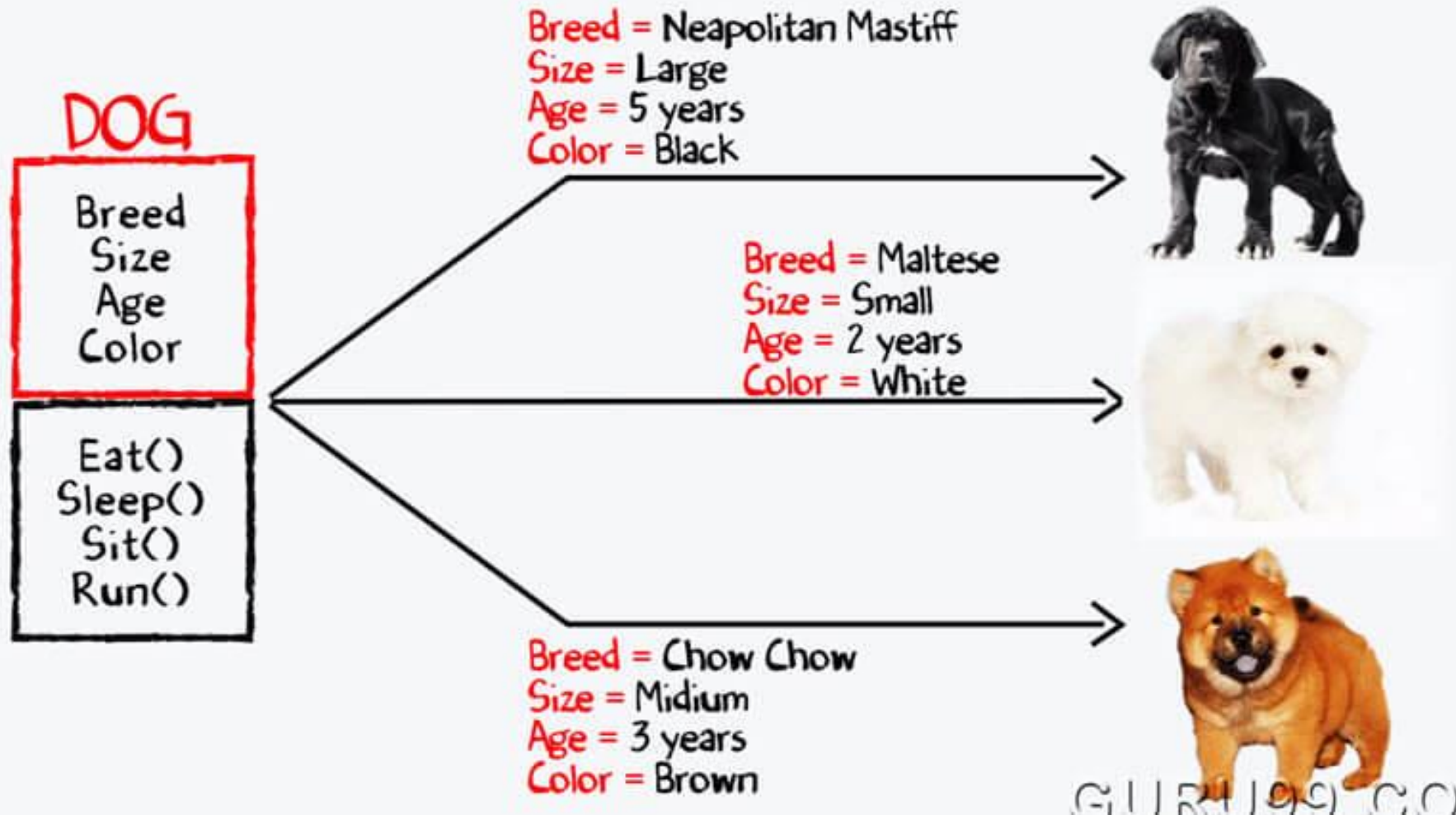
```
9     sleep(1000)
```




CLASSES & INSTANTIES I



CLASSES & INSTANCES II



CLASSES & INSTANCES III



Code
(procedures,
funcities,
methodes)

Data
(Sensor1,
Sensor2,
Sensor3)

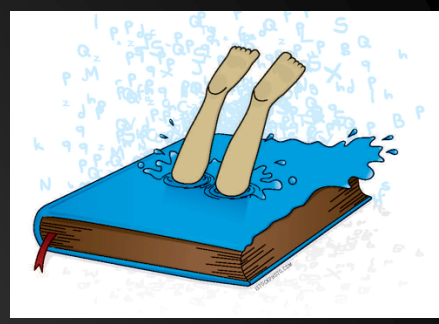
```
351 # create instance
352 Mq = MaqueenPlus()
```

CLASSES & INSTANCES

```
5 class GY53Pwm():
6
7     # -----
8     def __init__(self, InPin):
9         self.Distance = 9999
10        self.Pin        = InPin
11
12    # -----
13    def Read(self) :
14        while self.Pin.read_digi
15            pass # Avoid measurem
16
17        # High-time is proportio
```

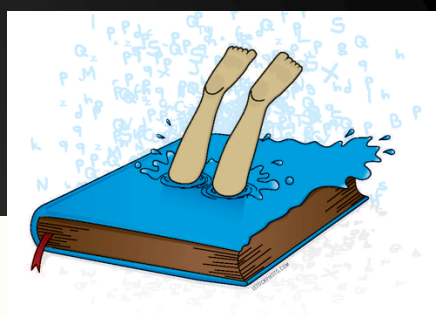
```
>>> GY53 = GY53Pwm(pin15)
>>>
>>> GY53.Distance
9999
>>> GY53.Read()
>>> GY53.Distance
419
>>>
```


CLASSES & INSTANCES

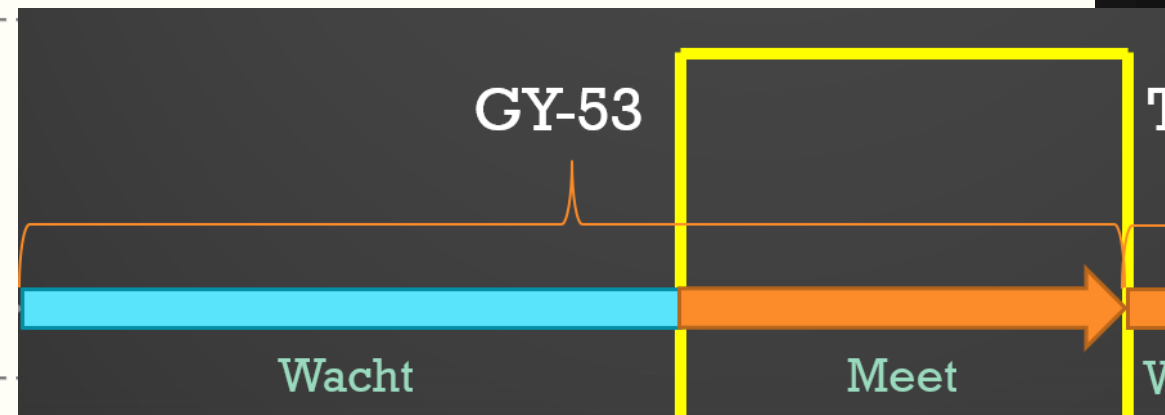


```
5 class GY53Pwm():
6
7     # -----
8     def __init__(self, InPin):
9         self.Distance = 9999
10        self.Pin = InPin
11
12    # -----
13    def Read(self) :
14        while self.Pin.read_digi
15            pass # Avoid measurem
16
17        # High-time is proportio
```

```
>>> GY53
<GY53Pwm object at 20005070>
>>>
>>> dir(GY53)
['__class__', '__init__', '__module__',
'__qualname__', '__dict__', 'Distance',
'Pin', 'Read']
>>>
>>> type(GY53.Distance)
<class 'int'>
>>>
```



```
5 class GY53Pwm():
6
7     # -----
8     def __init__(self, InPin):
9         self.Distance = 9999
10        self.Pin = InPin
11
12    # -----
13    def Read(self) :
14        while self.Pin.read_digital() == True:
15            pass # Avoid measurement errors: wait until input is low.
16
17        # High-time is proportional to distance, 10us = 1mm
18        # max 55 ms, pwm cycle time is 50ms
19        self.Distance = int(0.1 * time_pulse_us(self.Pin, 1, 55000))
```



TSENSORS

- Class voor VL53L0x en Ultrasoon samen.

```
273 class TSensors():
274
275     # -----
276     def __init__(self):
277         self.UsDistance      = 9999
278         self.GyDistance      = 9999
279         self.UsTriggerPin    = pin1
280         self.UsEchoPin       = pin2
281         self.GyPin           = pin15
282         self._UsCounter      = 99
283
284     # -----
285     def Takt(self) :
286         self._Ultrasonic()
287         self._GY53()
```

WANDVOLGEN – ERROR

```
21 # -----
22 def SequenceVolgWand(S):
23     if S.IsNewState('SequenceVolgWand') :
24         Mq.RGB(4, 4)
25
26     Correctie = (Sensoren.GyDistance - 570) * 0.4 # 500 mm afstand + 70 off
27     Correctie = max(-50, min(50, Correctie))
28
29     Mq.Motors(100 + Correctie, 100 - Correctie)
30     print("Afstand: %d, Correctie: %d" % (Sensoren.GyDistance, Correctie))
31
32     Mq.IsDone() # niet nodig voor Motors() maar kan geen kwaad (aanroep van
33
43 Sm.Goto([SequenceWachtA, SequenceVolgWand])
```

WANDVOLGEN – P-REGELAAR

```
21 # -----
22 def SequenceVolgWand(S):
23     if S.IsNewState('SequenceVolgWand') :
24         Mq.RGB(4, 4)
25
26     Correctie = (Sensoren.GyDistance - 570) * 0.4 # 500 mm afstand + 70 off
27     Correctie = max(-50, min(50, Correctie))
28
29     Mq.Motors(100 + Correctie, 100 - Correctie)
30     print("Afstand: %d, Correctie: %d" % (Sensoren.GyDistance, Correctie))
31
32     Mq.IsDone() # niet nodig voor Motors() maar kan geen kwaad (aanroep van
33
43 Sm.Goto([SequenceWachtA, SequenceVolgWand])
```

WANDVOLGEN – CLIP

```
21 # -----
22 def SequenceVolgWand(S):
23     if S.IsNewState('SequenceVolgWand') :
24         Mq.RGB(4, 4)
25
26     Correctie = (Sensoren.GyDistance - 570) * 0.4 # 500 mm afstand + 70 off
27     Correctie = max(-50, min(50, Correctie))
28
29     Mq.Motors(100 + Correctie, 100 - Correctie)
30     print("Afstand: %d, Correctie: %d" % (Sensoren.GyDistance, Correctie))
31
32     Mq.IsDone() # niet nodig voor Motors() maar kan geen kwaad (aanroep van
33
43 Sm.Goto([SequenceWachtA, SequenceVolgWand])
```

WANDVOLGEN – STURING

```
21 # -----
22 def SequenceVolgWand(S):
23     if S.IsNewState('SequenceVolgWand') :
24         Mq.RGB(4, 4)
25
26     Correctie = (Sensoren.GyDistance - 570) * 0.4 # 500 mm afstand + 70 off
27     Correctie = max(-50, min(50, Correctie))
28
29     Mq.Motors(100 + Correctie, 100 - Correctie)
30     print("Afstand: %d, Correctie: %d" % (Sensoren.GyDistance, Correctie))
31
32     Mq.IsDone() # niet nodig voor Motors() maar kan geen kwaad (aanroep van
33
43 Sm.Goto([SequenceWachtA, SequenceVolgWand])
```


WANDVOLGEN – STATEMACHINE

```
21 # -----
22 def SequenceVolgWand(S):
23     if S.IsNewState('SequenceVolgWand') :
24         Mq.RGB(4, 4)
25
26     Correctie = (Sensoren.GyDistance - 570) * 0.4 # 500 mm afstand + 70 off
27     Correctie = max(-50, min(50, Correctie))
28
29     Mq.Motors(100 + Correctie, 100 - Correctie)
30     print("Afstand: %d, Correctie: %d" % (Sensoren.GyDistance, Correctie))
31
32     Mq.IsDone() # niet nodig voor Motors() maar kan geen kwaad (aanroep van
33
43 Sm.Goto([SequenceWachtA, SequenceVolgWand])
```


OEFENING WANDVOLGEN

- Gebruik 3.wandvolgen.py

```
21 # -----
22 def SequenceVolgWand(S):
23     if S.IsNewState('SequenceVolgWand') :
24         Mq.RGB(4, 4)
25
26     Correctie = (Sensoren.GyDistance - 570) * 0.4 # 500 mm af
27     Correctie = max(-50, min(50, Correctie))
28
29     Mq.Motors(100 + Correctie, 100 - Correctie)
30     print("Afstand: %d, Correctie: %d" % (Sensoren.GyDistance
31
32     Mq.IsDone() # niet nodig voor Motors() maar kan geen kwaad
33
43 Sm.Goto([SequenceWachtA, SequenceVolgWand])
```

HUISWERK

- Maak wandvolgen werkend.
- Stop als minder dan 300mm van de achterwand bent.
- Geef de status van de regeling weer via de led's.

Bonuspunten:

- Maak wandvolgen beter.

Tip: gebruik een lange USB-kabel om met REPL te volgen wat geprint wordt.