

# Intro til robotteknologi

Med Raspberry Pi - PiARM

Daniel Matthiesen & Lasse Hadberg

TEC BALLERUP Telegrafvej 9

## Contents

Problemformulering .....	2
UML .....	2
PiARM Installationsvejledning .....	3
Udstyrsliste .....	3
Hardware .....	3
Software .....	3
Vedligeholdelse .....	3
Installation af PiARM controller & Servo Config .....	3
PiArm F.A.Q & Troubleshoot .....	4
Piarm kommandoerne eksekveres i modsat rækkefølge .....	4
Armen sidder løst .....	4
<i>Read</i> virker ikke i <i>PiArm Controller</i> .....	4
Hjælp! Jeg er kommet til at sætte parameters .....	4
Joystick Sourcekode .....	5
Dokumentation .....	11
PiArm Opgave .....	11
Update af raspberry pi .....	11
PiARM Controller .....	12
Servo Configuration .....	18
Draw opgave .....	23
Draw Sourcekode .....	24
Konklusion .....	24

## Problemformulering

I dette projekt vil vi gennemgå to mindre projekter. Ét der skal skabe et grundlag for din forståelse af *PiARM* samt dets tilhørende software. Det andet projekt hvori vi ved brug af Python kommunikere direkte til PiARM'en.

Introduktionsopgaven omhandler brug af *SB Components* egen software til at skabe forbindelse til, læse fra- og skrive til PiARM'ens servoer gennem en serielforbindelse. Ved brug af denne software tegner vi en firkant på et stykke papir.

I anden opgave vil vi direkte gennem pythonkode skabe forbindelse til, læse- og skrive til disse servoer. *SB Components* har leveret et eksempel på hvordan denne kode kan se ud. For en forbedret brugeroplevelse har vi valgt at raffinere deres kode og igennem denne revision har vi skabt en mere 'flydende' oplevelse.

## UML



# PiARM Installationsvejledning

## Udstyrsliste

### Hardware

1. Raspberry Pi
2. HDMI kabel
3. SD kort
4. Strømforsyning
5. Pc m. Skærm, mus & tastatur
6. Netkabel
7. PiARM: The DYI Robotic Arm for Raspberry Pi
8. Playstation controller m. trådløs dongle

### Software

1. Rasbian
2. Python
3. PiARM (Controller)
4. Servo Config

### Vedligeholdelse

- Lille stjerneskruetrækker
- 5.5 mm. Topnøgle

## Installation af PiARM controller & Servo Config

**OBS!! | OBS!! | OBS!! | OBS!! | OBS!! | OBS!! | OBS!!**

Hvis PiARM'en er samlet og tilsluttet Pi'en, Skal man IKKE ændre noget under "Parameters" inde i Servo Configuration. Dette vil sætte alle de tilsluttede Servo'er til samme ID.

1. Eksekvér følgende kode i terminalen:  
**git clone https://github.com/sbcshop/PiArm.git**
2. Filerne downloades og er derefter tilgængelige i PiArm mappen i */home/pi*
3. Skriv nu følgende i terminalen:  
**sudo chmod +x configGUI.py controlGUI.py**

## PiArm F.A.Q & Troubleshoot

Piarm kommandoerne eksekveres i modsat rækkefølge

Årsagen til dette er ukendt, men kan modvirkes med en genstartning af din Pi.

Armen sidder løst

Dette skyldes slidtage under bevægelse. Dette kan modvirkes ved brug af en stjerneskruestrækker samt en 5.5 mm. Topnøgle.

*Read* virker ikke i *PiArm Controller*

I tilfælde af, at værdierne ikke læses korrekt og man ønsker at bevæge ARM'en kan man implementere en løsning hvori man sætter ARM'ens placeringer i opstarten af ARM softwaren. Dette gør det muligt at bevæge ARM'en med de intervaller man har ønsket, hvorefter du kan udskrive hver servos nye position.

Hjælp! Jeg er kommet til at sætte parameters

Hvis du har sat parameters imens PiARM'en er sat til, vil alle de tilsluttede servo'ers ID blive sat til det samme ID. Dette kan løses ved at koble servo'erne fra hindenden en ad gangen, og så forbinde dem enkeltvis til Pi'en. Når de er forbundet uden de sidder sammen med andre, kan man gå ind under "Servo Configuration" i menuen "Parameters" og sætte servo'ens ID til det der står på servo'en. Efter alle er sat tilbage til det oprindelige ID kan PiARM'en blive samlet igen, og virke som den gjorde før.

Se under *Parameters* dokumentationen for hjælp til at sætte ID igen.

# Joystick Sourcecode

```
#!/usr/bin/python3
```

```
'''
```

This file contains Joystick example code for controlling PiArm  
Developed by - SB Components  
<http://sb-components.co.uk>

This project has been further optimized.  
Refined by - Daniel Matthiesen & Lasse Hadberg

```
'''
```

```
import re
import pygame
import piarm
from time import sleep
```

```
class Joystick_Controller(object):
```

```
    """
```

Joystick controller class

```
    """
```

```
def __init__(self):
```

```
    self.servo_POS_error = False
```

```
    self.joystick_keypress_status = False
```

```
    '''
```

self.step is used to define the value at which the servo will change its position.  
Increasing this will cause the position to change at larger intervals.  
This will in turn cause the controller to 'feel' less responsive.

```
    '''
```

```
    self.step = 5
```

```
    # This is used to define the initial position of the ARM.
```

```
    self.servo_position = {
```

```
        1: 500,
```

```
        2: 500,
```

```
        3: 500,
```

```
        4: 500,
```

```
        5: 500,
```

```
        6: 500,
```

```
    }
```

```
    # Button Status is used to tell wether a button is pressed or not.
```

```
    self.button_status = {
```

```
        0: 0,
```

```
        1: 0,
```

```
        2: 0,
```

```
        3: 0,
```

```
        4: 0,
```

```
        5: 0,
```

```
        6: 0,
```

```
    }
```

```
    # Initialize Joystick
```

```
    pygame.init()
```

```

try:
    self.controller = pygame.joystick.Joystick(0)
    self.controller.init()
    print('Joystick initialized')
except pygame.error as pygame_err:
    print(pygame_err)
# This initial read of the servo positions allows us to determine if the positions
are read successfully.
self.read_servo_position()
print(self.servo_position)

```

```
def listen(self):
```

```
    """
```

```
    Listen for events to happen
```

```
    """
```

```
    while True:
```

```
        for event in pygame.event.get():
```

```
            # Button Pressed
```

```
            if event.type == pygame.JOYBUTTONDOWN:
```

```
                # set Servo to default - Start Button
```

```
                if event.button == 9:
```

```
                    done = 0
```

```
                    servoid = 1
```

```
                    while done == 0:
```

```
                        print("Servo number { " + str(servoid) + " } has been reset to
```

```
default")
```

```
                        self.servo_position[servoid] = 500
```

```
                        robot.servoWrite(servoid, self.servo_position[servoid], 500)
```

```
                        servoid = servoid+1
```

```
                        if servoid == 7:
```

```
                            done = 1
```

```
                    print("Alle Servoer er returneret til default position")
```

```
                    self.servo_position = {
```

```
                        1: 500,
```

```
                        2: 500,
```

```
                        3: 500,
```

```
                        4: 500,
```

```
                        5: 500,
```

```
                        6: 500,
```

```
                    }
```

```
    """
```

This sleep is necessary as the movement to default position takes 0.5 seconds.

If left out, changing the position prematurely is possible and can cause placement issues.

```
    """
```

```
    sleep(0.5)
```

```
    # keypress
```

```
    elif event.button in range(13):
```

```
        self.button_status[event.button] = 1
```

```
        print('button { } pressed: '.format(event.button),
```

```
              self.button_status[event.button])
```

```
    print(self.button_status)
```

```

# Button Released
elif event.type == pygame.JOYBUTTONDOWN:
    if event.button in range(13):
        self.button_status[event.button] = 0
        print('button {} released: '.format(event.button),
              self.button_status[event.button])

# If facing buttons are pressed reset button position
if self.button_status[0] == self.button_status[2]:
    self.button_status[0] = 0
    self.button_status[2] = 0
if self.button_status[1] == self.button_status[3]:
    self.button_status[1] = 0
    self.button_status[3] = 0

# Change servo position
"""
As the state of button_status changes, the input from the controller will
determine which
'button' is pressed and as such which command to execute.
"""
for button, status in self.button_status.items():
    if status:
        # Move UP - Triangle
        if button == 0:
            if self.servo_position[5] - self.step in range(500, 900):
                self.servo_position[5] -= self.step
            if self.servo_position[4] + self.step in range(215, 500):
                self.servo_position[4] += (self.step + 10)
            if self.servo_position[3] - self.step in range(500, 880):
                self.servo_position[3] -= (self.step + 15)

        # Move Left - Square
        elif button == 1:
            if self.servo_position[6] + self.step in range(70, 900):
                self.servo_position[6] += self.step * 10

        # Move Down - Cross
        elif button == 2:
            if self.servo_position[5] + self.step in range(500, 595):
                self.servo_position[5] += self.step
            if self.servo_position[4] - self.step in range(215, 500):
                self.servo_position[4] -= (self.step + 10)
            if self.servo_position[3] + self.step in range(500, 880):
                self.servo_position[3] += (self.step + 15)

        # Move Right - Circle
        elif button == 3:
            if self.servo_position[6] - self.step in range(125, 951):
                self.servo_position[6] -= self.step * 10

        # Claw Open - L1
        elif button == 4:

```



```

        if self.servo_position[1] - self.step in range(160, 800):
            self.servo_position[1] -= (self.step * 15)

# Claw Close - R1
elif button == 5:
    if self.servo_position[1] + self.step in range(30, 710):
        self.servo_position[1] += (self.step * 15)

# Claw Left - L2
elif button == 6:
    if self.servo_position[2] + self.step in range(125, 875):
        self.servo_position[2] += (self.step * 15)

# Claw Right - R2
elif button == 7:
    if self.servo_position[2] - self.step in range(125, 875):
        self.servo_position[2] -= (self.step * 15)

# Arm Retract Arm - R3
elif button == 10:
    if self.servo_position[5] - self.step in range(500, 750):
        self.servo_position[5] -= self.step
    if self.servo_position[4] - self.step in range(215, 500):
        self.servo_position[4] -= (self.step + 10)
    if self.servo_position[3] + self.step in range(500, 880):
        self.servo_position[3] += (self.step + 15)

# Arm Extend Arm - L3
elif button == 11:
    if self.servo_position[5] + self.step in range(500, 750):
        self.servo_position[5] += self.step
    if self.servo_position[4] + self.step in range(215, 500):
        self.servo_position[4] += (self.step + 10)
    if self.servo_position[3] - self.step in range(500, 880):
        self.servo_position[3] -= (self.step + 15)

# The keypress status is changed to true, this tells the program that it's
time to move a servo.
self.joystick_keypress_status = True

# Write current positions
if self.joystick_keypress_status:
    if robot.alive:
        for ID in range(1, 7):
            robot.servoWrite(ID, self.servo_position[ID], 250)
            print("Writing to servo Id { } position: ".format(ID),
self.servo_position[ID])
            self.joystick_keypress_status = False
    else:
        print('Comm port is not connected')
'''
# For a fluid movement, it's important that this sleep is synchronized with
the
movement timer for the servoWrite.

```

```

        """
        sleep(0.25)

def read_servo_position(self):
    """
    This function read current servo position
    """
    if robot.alive:
        try:
            # Read Positions of motors one at a time
            for ID in range(1, 7):
                done = 0
                while done == 0:
                    postrue = 0
                    response = robot.positionRead(ID)

                    for element in response:
                        postrue = postrue + 1

                    if postrue >= 7:
                        pos = int.from_bytes(response[5]+response[6], byteorder='little')
                        # Button Position to variable
                        self.servo_position[ID] = pos
                        done = 1

                if self.servo_POS_error:
                    print("Servo Error", "Servo " + str(ID) +
                        ' - Position Out of Range..!')
                else:
                    print("Motor position Read Done Successfully")

            except TypeError:
                print("Servo Error", "Servo " + str(ID) +
                    ' - Not Responding')

if __name__ == "__main__":
    robot = piarm.PiArm()
    # write your serial comm.
    robot.connect("/dev/ttyUSB0")
    joystick = Joystick_Controller()
    # Start Joystick
    try:
        joystick.listen()
    # Set Motors to Default at KeyboardInterrupt
    except KeyboardInterrupt:
        pass
    for ID in range(1, 7):
        robot.servoWrite(ID, 500, 1500)

```

Nedenfor ses terminalens output ved opstart af programmet. I tredje række ses joystick initialiseringen efterfulgt af en servoerror. Denne er en enkelt læsning hvor vi ej har fået respons fra en servo. Dette rettes på på senere som vi kan se da servoernes pladsering successfult bliver sat til default. I række 5 ses det, at første- til fjerde servos værdier er uden for vores range (0-1000). For at vi har kunne få denne indenfor en operable range skriver vi den ønskede position til hver servo.

```
Running: joystick.py
pygame 1.9.4.post1
Hello from the pygame community. https://www.pygame.org/contribute.html
Joystick initialized
Servo Error Servo 5 - Not Responding
(1: 65473, 2: 25076, 3: 6643, 4: 61311, 5: 500, 6: 500)
Servo number (1) has been reset to default
Servo number (2) has been reset to default
Servo number (3) has been reset to default
Servo number (4) has been reset to default
Servo number (5) has been reset to default
Servo number (6) has been reset to default
Alle Servoer er returneret til default position
{0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0}
button 9 released: 0
```

Nedenfor ser vi udskrivninger vi ønsker efter hver tast på en knap. Vi udskriver servoernes state (Om de trykkes eller ej) og hvilken position disse bliver skrevet til. Ligeledes skrives der hvilken knap der er blevet trykket, og hvilken der er blevet frigivet.

```
Writing to servo Id 2 position: 350
Writing to servo Id 3 position: 680
Writing to servo Id 4 position: 365
Writing to servo Id 5 position: 545
Writing to servo Id 6 position: 350
button 7 released: 0
button 2 pressed: 1
{0: 0, 1: 0, 2: 1, 3: 0, 4: 0, 5: 0, 6: 0, 9: 0, 7: 0}
Writing to servo Id 1 position: 650
Writing to servo Id 2 position: 350
Writing to servo Id 3 position: 700
Writing to servo Id 4 position: 350
Writing to servo Id 5 position: 550
Writing to servo Id 6 position: 350
button 2 released: 0
```

# Dokumentation

## PiArm Opgave

### Update af raspberry pi

Hvis der er brug for at raspberry pi'en bliver opdateret gøres det som følgende.

Åben terminalen.



Derefter indtast: `pi@raspberrypi:~ $ sudo apt update`

Og lad det køre færdigt

Til sidst skal: `pi@raspberrypi:~ $ sudo apt full-upgrade`

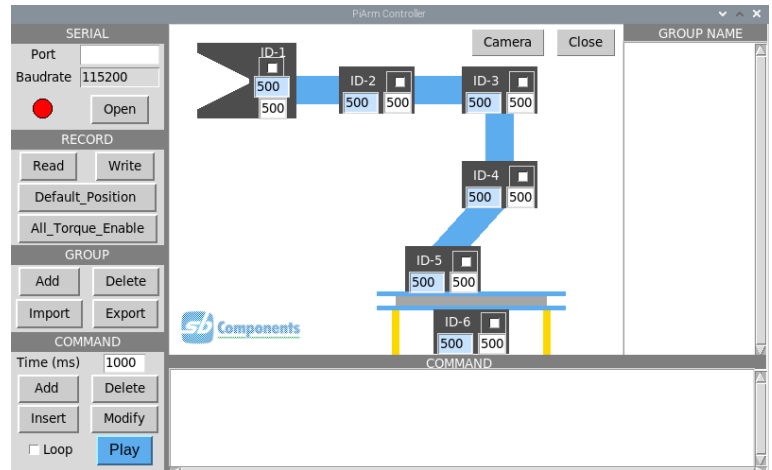
Indtasts og lad den opdatere.

```
pi@raspberrypi:~ $ sudo apt update
Get:1 http://archive.raspberrypi.org/debian buster InRelease [25.2 kB]
Get:2 http://raspbian.raspberrypi.org/raspbian buster InRelease [15.0 kB]
Get:3 http://raspbian.raspberrypi.org/raspbian buster/main armhf Packages [13.0 MB]
Get:4 http://archive.raspberrypi.org/debian buster/main armhf Packages [259 kB]
Fetched 13.3 MB in 26s (509 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
10 packages can be upgraded. Run 'apt list --upgradable' to see them.
pi@raspberrypi:~ $ sudo apt full-upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  file libmagic-mgc libmagic1 openjdk-11-jdk openjdk-11-jdk-headless
  openjdk-11-jre openjdk-11-jre-headless raspi-config rpi-eeeprom
  rpi-eeeprom-images
10 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 219 MB of archives.
After this operation, 9,092 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
```

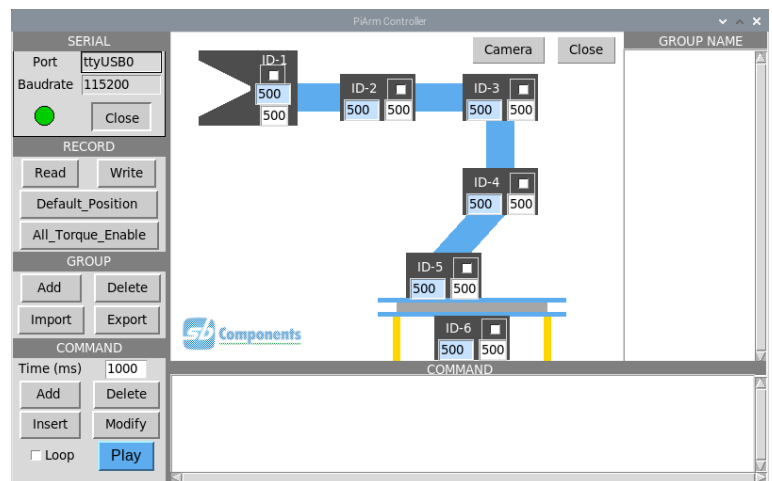
## PiARM Controller

### Forbind til PiARM Controller

Når man åbner programmet kommer dette vindue frem.



For at forbinde til armen skal port navnet på den port armen er forbundet til indtastes. Derefter skal der trykkes på *open* knappen.

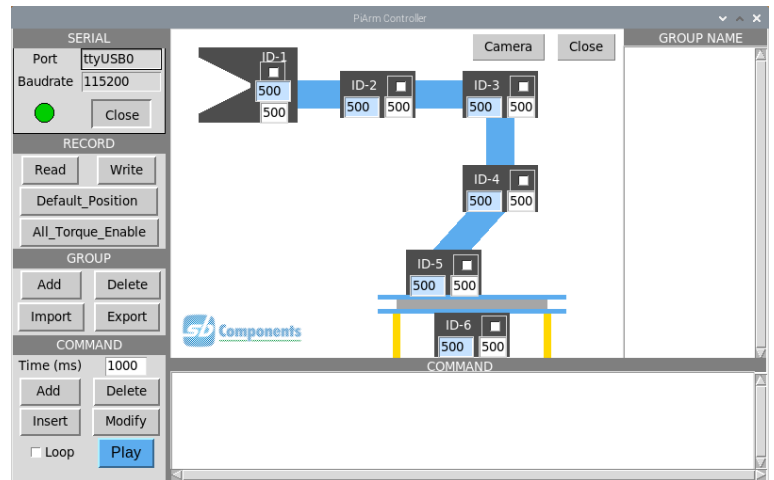


### Brugbare knapper

Ved at trykke på *Default\_Position* vil PiARM'en blive at i dens standard position som er værdi 500.

Ved at trykke på *All\_Torque\_Enable* vil alle servo'er blive løsnnet, og dermed kan armen blive rykket frit.

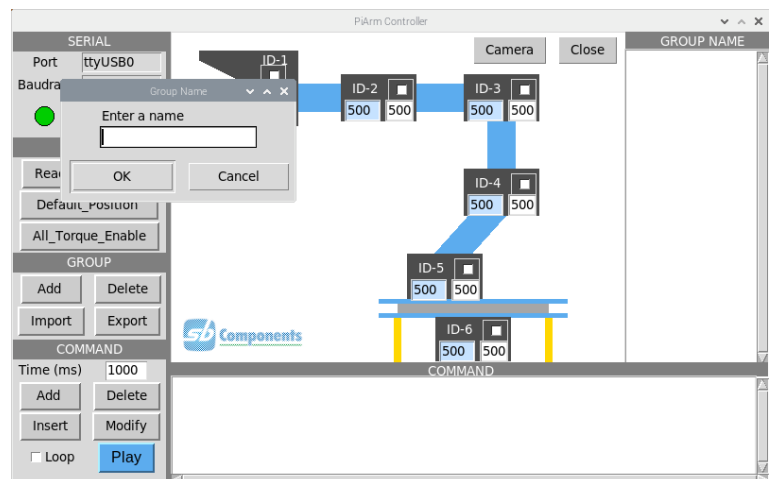
Når knappen trykkes igen strammes alle servo'er igen så de bliver i samme positioner.



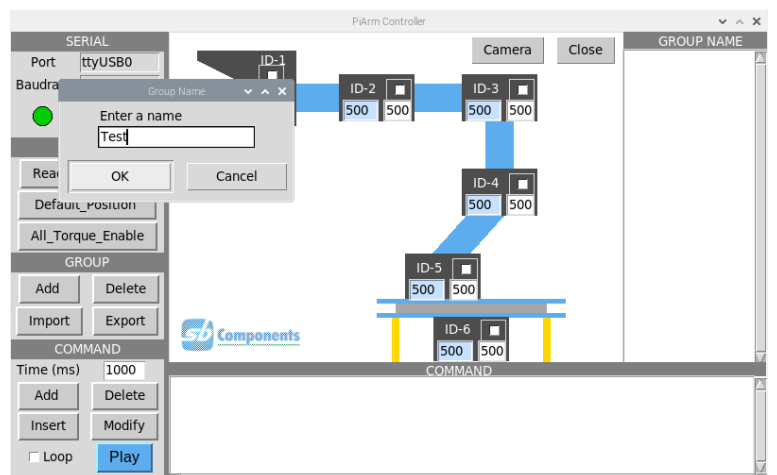
### Opretning af gruppe

Opret en gruppe for at kunne lave en sekvens af bevægelser.

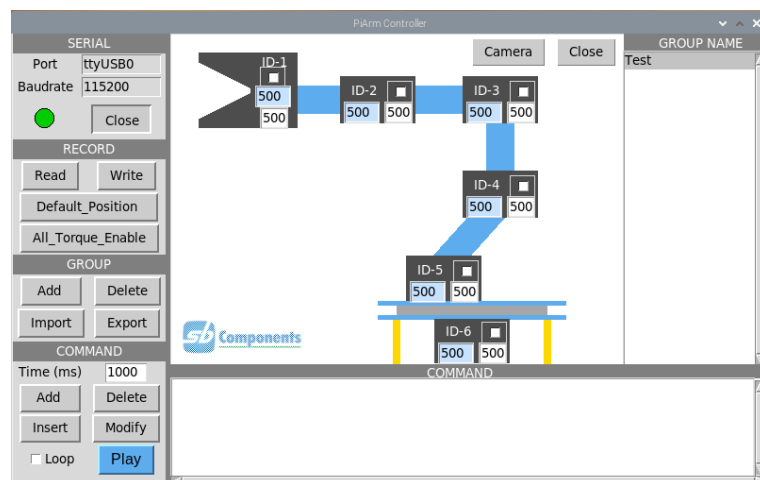
Tryk på *Add* under *GROUP* menuen og så kommer der en boks frem hvor gruppens skrives.



Efter at have indtastet navnet på gruppen, tryk *ok* og gruppen bliver oprettet.



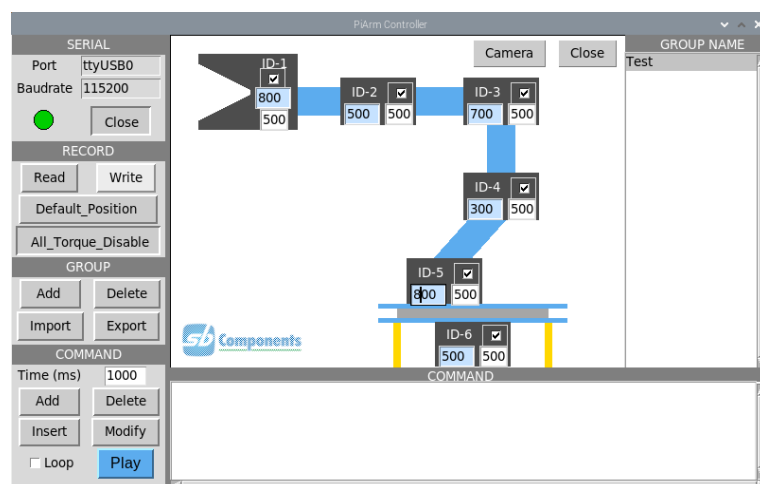
Når *ok* er trykket kommer gruppen frem i kolonnen og tilføjelse af positioner kan begyndes.



### *Tilføjelse af armens position til en gruppe*

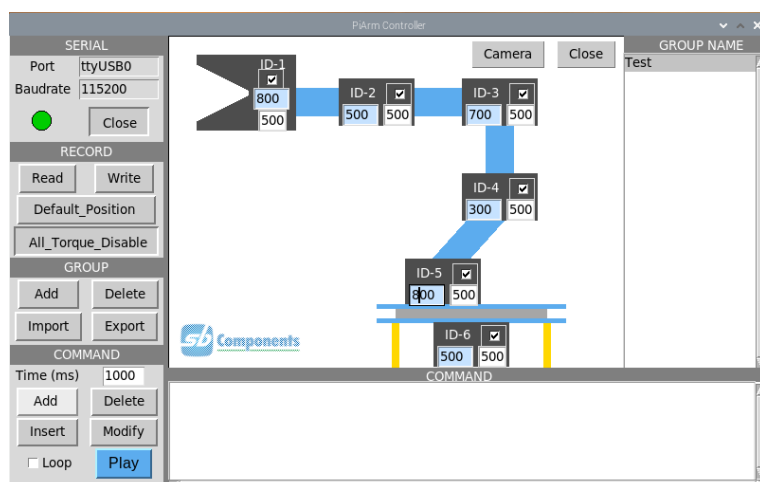
For at tilføje positioner skal tal indtastes ud for servo'ernes ID.

Det blå felt er positionen servo'en ændres til, det hvide felt er hvor lang tid den er om at ændre til den position i millisekunder.



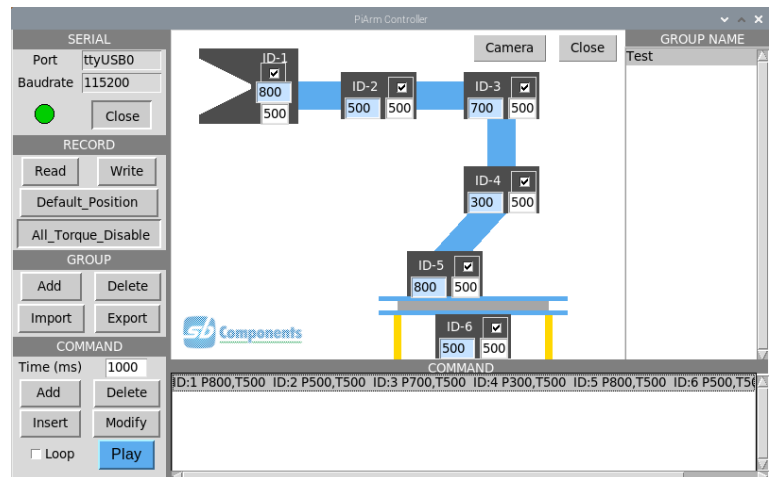
Efter at have indtastet de ønskede tal er indtastet, tryk på *Add* under *COMMAND* menu'en for at tilføje den til gruppen.

Tallet ud for *Time* er tiden i millisekunder som der går før den begynder at gå til næste position.

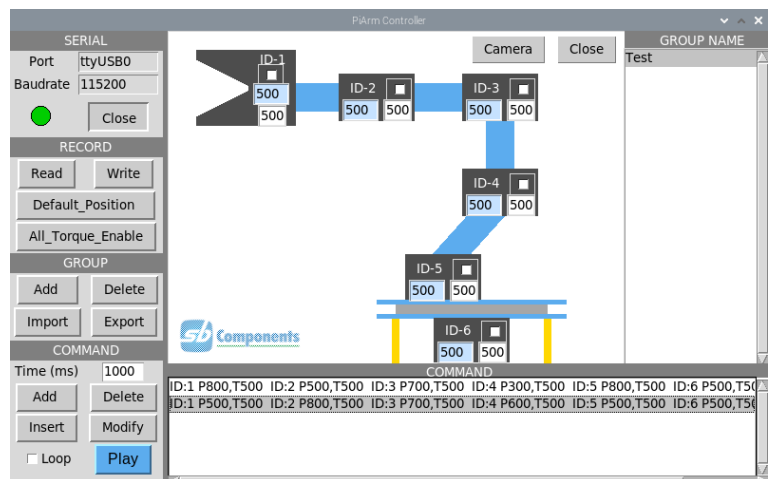


Efter at have trykket ”Add” bliver positionen tilføjet i bunden under *COMMAND*.

ved er dobbelt klikke på en position under *COMMAND* vil PiARM’en bevæge sig til den position.

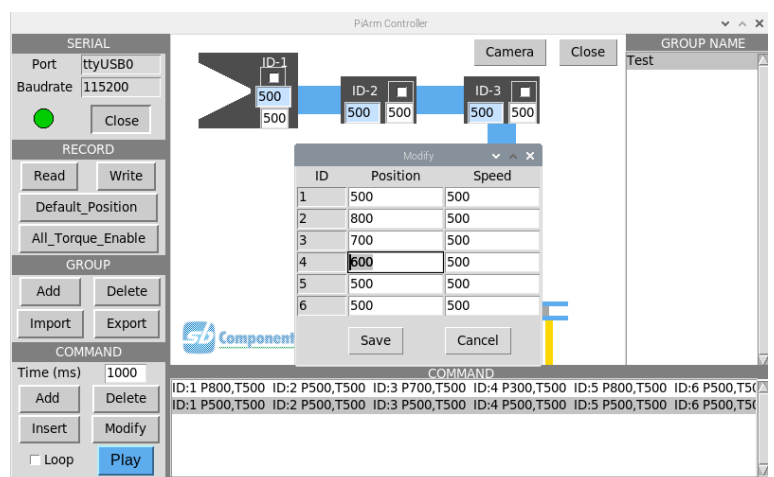


Efter at have valgt en position under *COMMAND* kan *Modify* trykkes for at ændre på en allerede tilføjet position.



Når *Modify* er trykket kommer der en boks frem med de nuværende værdier. Ændre på de værdier der skal ændres og tryk *Save* for at gemme.

Det er også muligt at ændre i filen (/home/pi/PiArm/Export Files).



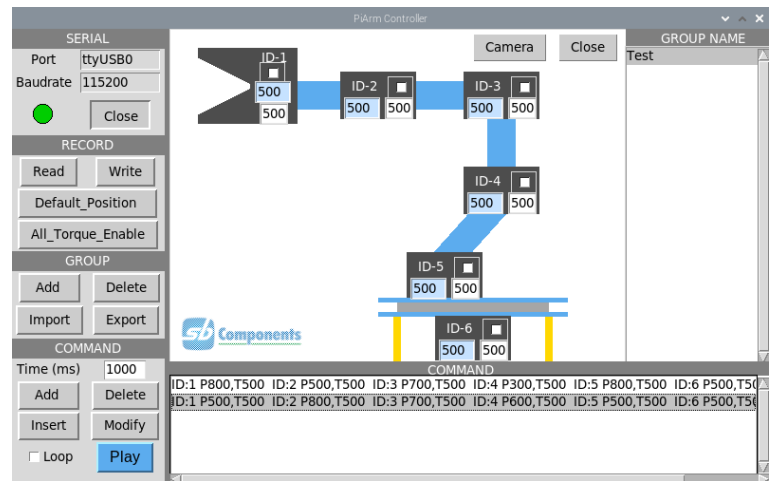


### Afspilning af sekvens

Når en gruppe er valgt kan *Play* trykkes og gruppens sekvens bliver afspillet.

Mens sekvensen er i gang kan *Stop* trykkes for at stoppe igen.

Ved at sætte flueben ved *Loop* vil sekvensen blive afspillet i et loop.

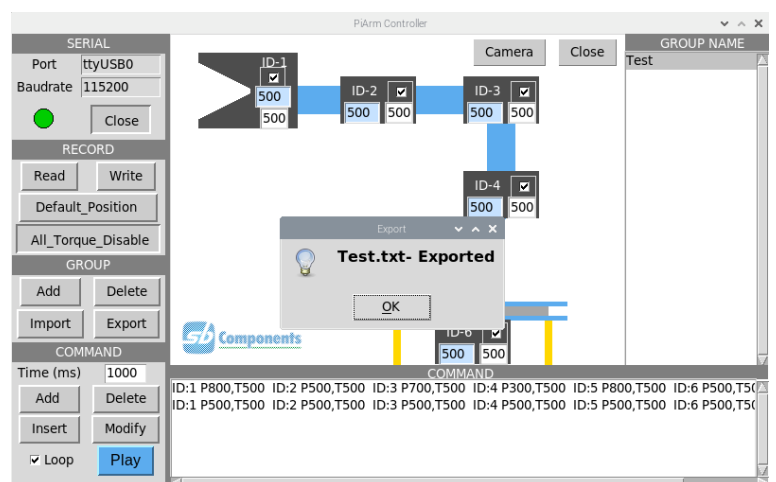


### Export af en gruppe

For at gemme en gruppe skal den exporteres.

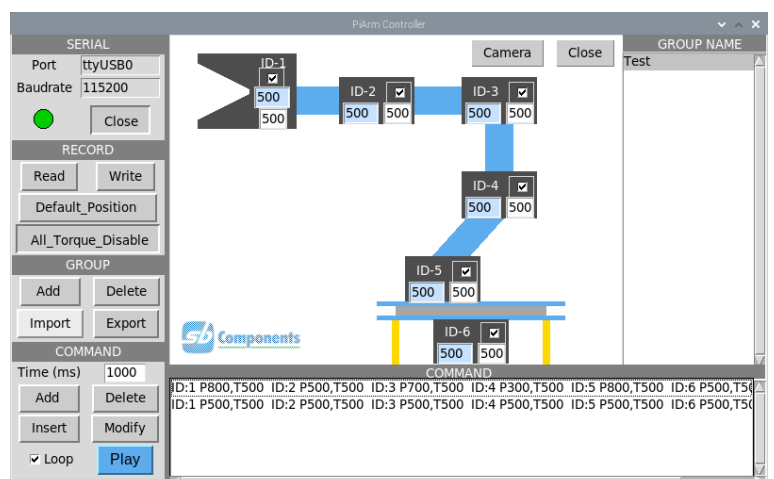
Dette sker ved at have valgt den ønskede gruppe i højre kolonne, og derefter trykke på *Export* under *GROUP* menu'en.

Når dette er gjort kommer en boks frem som fortæller at exporteringen er fuldført.



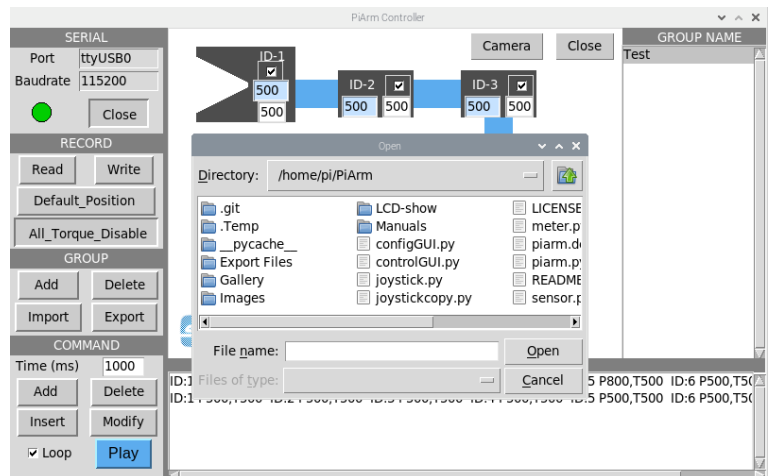
### Import af gruppe

For at hente en gemt gruppe kan *Import* trykkes under *GROUP* menu'en.



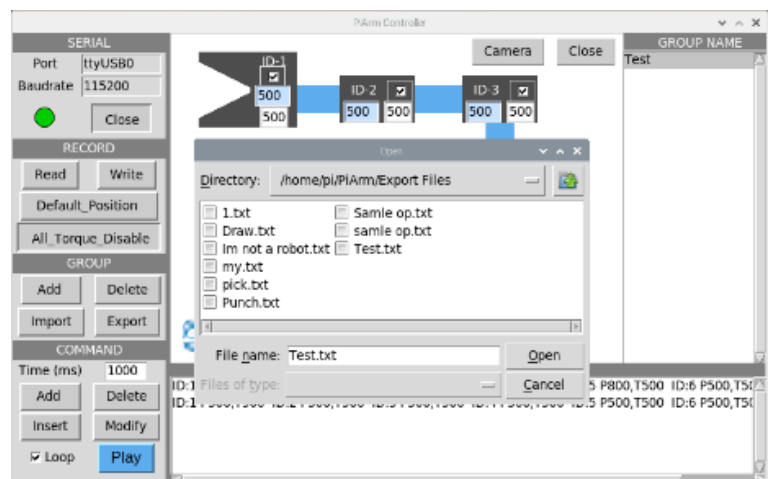
En mappe kommer frem med alle filer der tilhører PiARM'en.

Vælg *Export Files* for at vise de exporterede grupper.

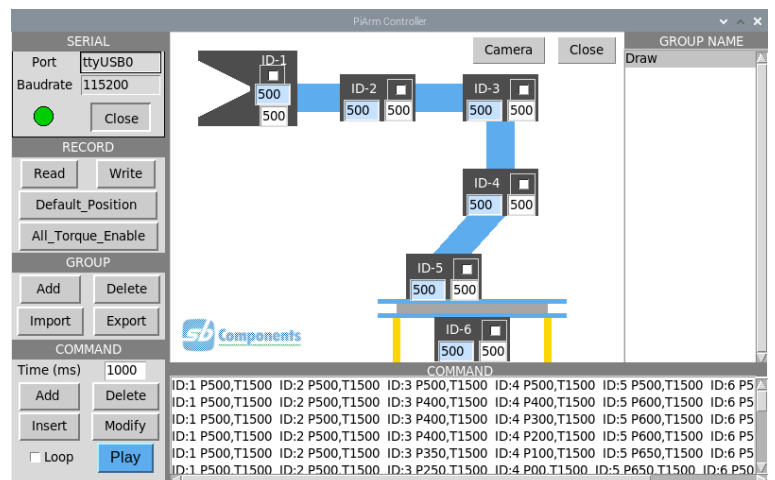


Når dette er gjort kommer en mappe frem hvor alle exporterede grupper ligger.

Vælg den gruppe der skal åbnes og tryk *Open*.



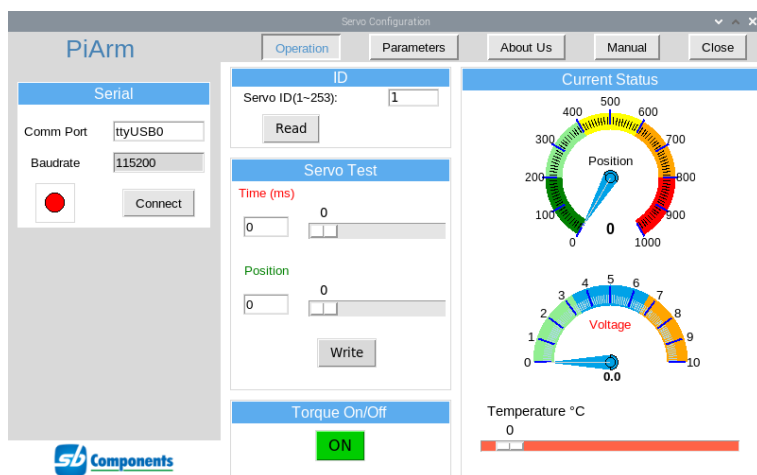
Gruppen kan nu ses i højre kolonne og er klar til at blive brugt.



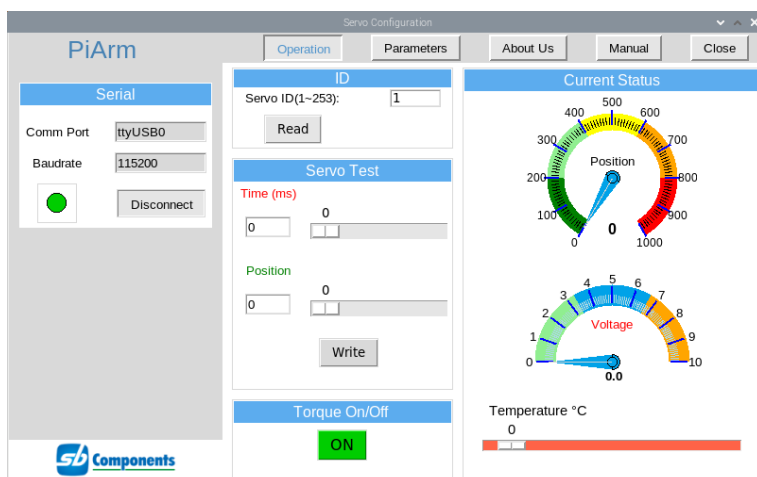
## Servo Configuration

### Forbind til Servo Configuration

Når man åbner programmet kommer dette vindue frem. Tryk *Connect* for at forbinde til PiArm'en.



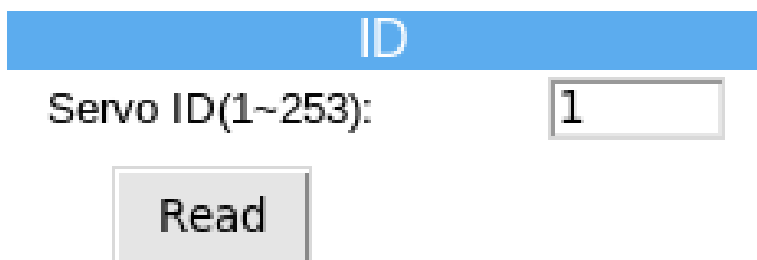
Derefter kan servo'erne ændres på.



### Operation

Under *Servo ID* skal ID'et på den servo der skal ændres på indtastes.

Med *Read* kan man læse værdierne på servo'en, dog kan der være problemer med denne funktion



*Servo Test* er hvor ser kan ændres på en servo's position og hastigheden på den.

*Time* er hvor hastigheden på servo'en ændres.

*Position* er hvor servo'ens position ændres.

Der kan både bruges slideren eller skrive et tal i feltet.

*Write* trykkes når værdierne skal skrives til Servo'en.

*Torque* er en funktion som gør servo'en løs eller strammer den.

Den ændes ved at trykke *ON* og *OFF*.

Hvis *Read* fungerer kan data servo'ens værdier aflæses her.

*Position* er servo'ens position.

*Voltage* er hvor mange volt som servo'en bruger.

*Temperature* er hvor varm servo'en er.

## Servo Test

Time (ms)

0



Position

0

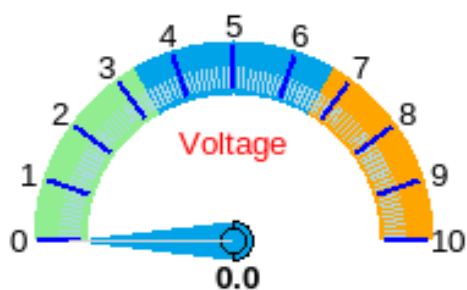
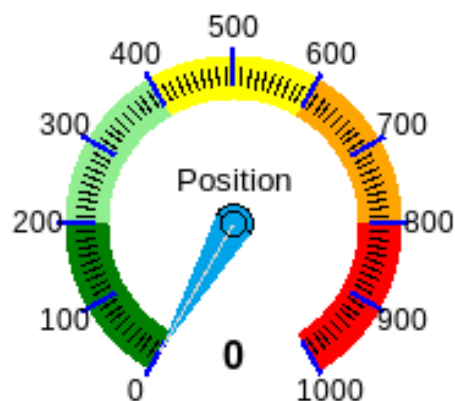


Write

## Torque On/Off

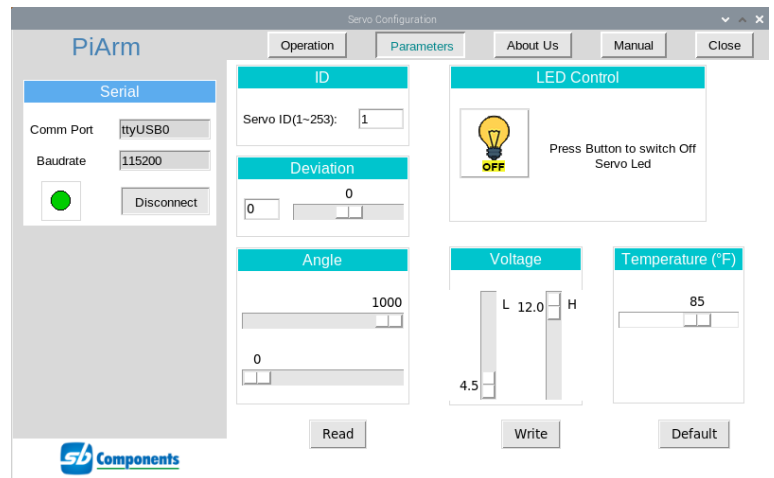
ON

## Current Status



## Parameters

Under *parameters* kan man indstille servo'er.



*ID* bruges til at sætte ID på de servo'er der er tilsluttet.

Vær opmærksom på at hvis der er mere end en servo tilsluttet, vil de få samme ID.



Servo ID(1~253):

Ved *Deviation* kan man rette servo'en til så den står lige.

Dette kan både gøre med slideren eller ved at indtaste et tal i feltet til venstre.



Under *Angle* kan man ændre den vinkel som servo'en skal kunne bevæge sig inden for.

Den øverste slider er den øverste grænse.

Den nederste slider er den nederste grænse.

## Angle

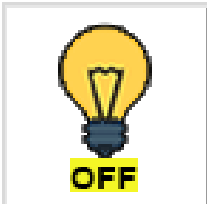
1000

0

---

Ved *LED Control* kan man tænde og slukke for LED'en på servo'en.

## LED Control



Press Button to switch Off  
Servo Led

Ved *Voltage* kan den øvre og nedre grænse sættes for hvor mange volt servo'en må bruge.

Den venstre slider er den nederste grænse.

Den højre slider er den øverste grænse.

## Voltage



Under *Temperature* kan man sætte den en grænse på hvor varm servo'en må være før den slår fra.

Tempearturen er i fahrenheit.

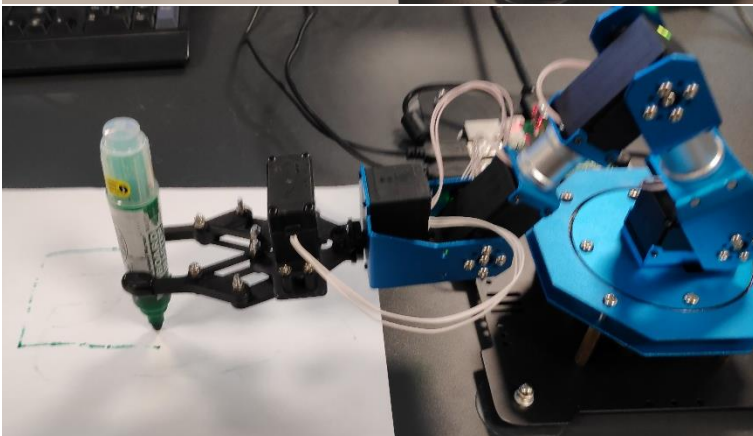
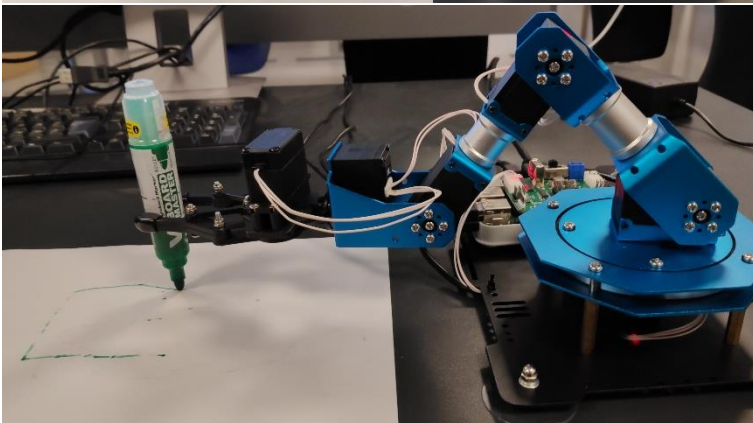
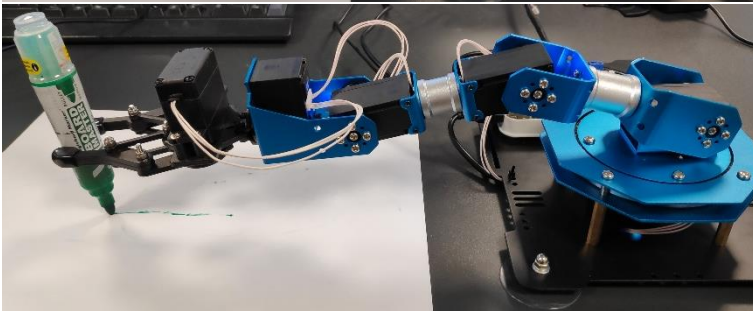
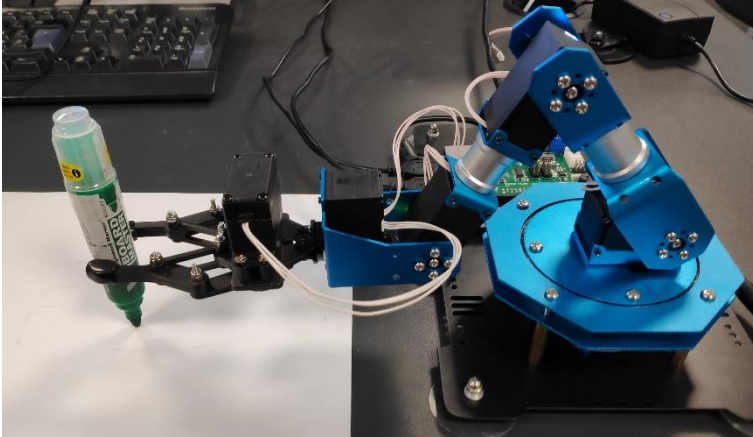
## Temperature (°F)



## Draw opgave

I denne opgave skulle vi lave en sekvens som kunne tegne en firkant.

Det er blevet lavet igennem *PiARM Controller* som er *SB Components* software.





## Draw Sourcekode

```
ID:1 P500,T1500 ID:2 P500,T1500 ID:3 P500,T1500 ID:4 P500,T1500 ID:5 P500,T1500 ID:6 P500,T1500 Time:1500
ID:1 P500,T1500 ID:2 P500,T1500 ID:3 P400,T1500 ID:4 P400,T1500 ID:5 P600,T1500 ID:6 P500,T1500 Time:1500
ID:1 P500,T1500 ID:2 P500,T1500 ID:3 P400,T1500 ID:4 P300,T1500 ID:5 P600,T1500 ID:6 P500,T1500 Time:1500
ID:1 P500,T1500 ID:2 P500,T1500 ID:3 P400,T1500 ID:4 P200,T1500 ID:5 P600,T1500 ID:6 P500,T1500 Time:1500
ID:1 P500,T1500 ID:2 P500,T1500 ID:3 P350,T1500 ID:4 P100,T1500 ID:5 P650,T1500 ID:6 P500,T1500 Time:1500
ID:1 P500,T1500 ID:2 P500,T1500 ID:3 P250,T1500 ID:4 P0,T1500 ID:5 P650,T1500 ID:6 P500,T1500 Time:1500
ID:1 P600,T1500 ID:2 P500,T1500 ID:3 P250,T1500 ID:4 P0,T1500 ID:5 P650,T1500 ID:6 P500,T1500 Time:1500
ID:1 P700,T1500 ID:2 P500,T1500 ID:3 P250,T1500 ID:4 P0,T1500 ID:5 P650,T1500 ID:6 P500,T1500 Time:1500
ID:1 P700,T1500 ID:2 P500,T1500 ID:3 P250,T1500 ID:4 P0,T1500 ID:5 P650,T1500 ID:6 P500,T1500 Time:1500
ID:1 P700,T1500 ID:2 P500,T1500 ID:3 P330,T1500 ID:4 P100,T1500 ID:5 P680,T1500 ID:6 P500,T1500 Time:1500
ID:1 P700,T1500 ID:2 P500,T1500 ID:3 P365,T1500 ID:4 P150,T1500 ID:5 P700,T1500 ID:6 P500,T1500 Time:1500
ID:1 P700,T1500 ID:2 P500,T1500 ID:3 P470,T1500 ID:4 P350,T1500 ID:5 P800,T1500 ID:6 P500,T1500 Time:1500
ID:1 P700,T1500 ID:2 P500,T1500 ID:3 P500,T1500 ID:4 P450,T1500 ID:5 P850,T1500 ID:6 P450,T1500 Time:1500
ID:1 P700,T1500 ID:2 P500,T1500 ID:3 P370,T1500 ID:4 P150,T1500 ID:5 P700,T1500 ID:6 P440,T1500 Time:1500
ID:1 P700,T1500 ID:2 P500,T1500 ID:3 P280,T1500 ID:4 P50,T1500 ID:5 P670,T1500 ID:6 P440,T1500 Time:1500
ID:1 P700,T1500 ID:2 P500,T1500 ID:3 P310,T1500 ID:4 P10,T1500 ID:5 P630,T1500 ID:6 P460,T1500 Time:1500
ID:1 P700,T1500 ID:2 P500,T1500 ID:3 P280,T1500 ID:4 P0,T1500 ID:5 P650,T1500 ID:6 P500,T1500 Time:1500
ID:1 P700,T1500 ID:2 P500,T1500 ID:3 P250,T1500 ID:4 P0,T1500 ID:5 P650,T1500 ID:6 P500,T1500 Time:1500
ID:1 P700,T1500 ID:2 P500,T1500 ID:3 P250,T1500 ID:4 P0,T1500 ID:5 P650,T1500 ID:6 P500,T1500 Time:1500
ID:1 P600,T1500 ID:2 P500,T1500 ID:3 P250,T1500 ID:4 P0,T1500 ID:5 P650,T1500 ID:6 P500,T1500 Time:1500
ID:1 P500,T1500 ID:2 P500,T1500 ID:3 P250,T1500 ID:4 P0,T1500 ID:5 P650,T1500 ID:6 P500,T1500 Time:1500
ID:1 P500,T1500 ID:2 P500,T1500 ID:3 P350,T1500 ID:4 P200,T1500 ID:5 P550,T1500 ID:6 P500,T1500 Time:1500
ID:1 P500,T1500 ID:2 P500,T1500 ID:3 P500,T1500 ID:4 P500,T1500 ID:5 P500,T1500 ID:6 P500,T1500 Time:1500
```

## Konklusion

Efter en længere fejlfindingsproces har vi erfaret at der i vores software findes en fejl der gør, at læsningerne fra både *SB Components*' software samt vores egen har været forstyrrede. Dette har gjort, at *read* funktionaliteten i førstnævntes software har givet fejl. Da denne egenskab ikke har været afgørende for vores usecase har vi udviklet en metode der afhjælper problemet. Dette ses i pythonkoden under tast '9' (start-knappen) hvor vi til at starte med sætter alle servoers position til en valgt statisk position.

Da vi udviklede løsningen til tegnefunktionen, har vi ikke mødt disse læseproblemer. *SB Components* egen software er let forståeligt. Vi har derved oplevet tegneprocessen som utroligt let.

Arbejdet med ARM'en har foruden førstnævnte fejl været 'let'. Dette er grundet *SB Components* inklusion af et kodeeksempel. Dette har vi videreudviklet med yderligere funktionalitet for at skabe en mere 'flydende' oplevelse. ARM'en har efter denne revision flere skrivningspunkter. Dette gør, at dennes bevægelser virker i højere grad responsiv.