# TESTPLAN

**Auteur:** Ivo Bruinsma

**Inlevermoment:** 50% oplever-set

**Docenten:** Anne de Gier & Wouter Volders

4MT
Nijverheidstraat 10
2751GR Moerkapelle
079-5932121
www.4mt.nl

# Revisiebeheer

| Revisie | Auteur | Datum | Wijziging |
|---|---|---|---|
| | Ivo Bruinsma | 18-09-2025 | Aanmaak document |
| | Ivo Bruinsma | 15-10-2025 | Invoegen nep hortensia & Autonome test |
| | Ivo Bruinsma | 27-10-2025 | Invoegen grijper test |
| | Ivo Bruinsma | 27-10-2025 | Invullen hortensia & Autonome test |
| | Ivo Bruinsma | 30-10-2025 | Invullen grijper v0.3 & v1.3 grijper test |
| | Ivo Bruinsma | 3-11-2025 | Invoegen & invoeren controller input test |
| | Ivo Bruinsma | 10-11-2025 | Veiligheid test invoegen en uitwerken |
| | Ivo Bruinsma | 13-11-2025 | Noodknop controller test invoegen |

Bij het maken van een nieuwe versie, eerst kopie maken van oude versie en onder nieuwe bestandsnaam opslaan. Dan versienummers in document aanpassen en revisiebeheer invullen.

Voordat wijzigingen gemaakt kunnen worden, dienen de vorige wijzigingen eerst geaccepteerd te worden (Controleren -> Accepteren -> Alle wijzigingen accepteren). Zet vervolgens ‘Wijzigingen bijhouden’ aan.

# Inhoudsopgave

# Nep hortensia verplaatsen

In deze test wordt er getest waar de hortensia's het beste opgepakt kunnen worden.

## User story

Met deze test worden user story 2 & 4 uit de originele backlog getest.

## Testopstelling en benodigdheden

De testopstelling is als volgt. De Fairino FR5 is gemonteerd op een kleine pallet met een houten plank op de manier zoals beschreven is in de instructies van Fairino. Deze opstelling staat in een ruimte waarbij de FR5 vrijuit kan bewegen. De hortensia welke in de test gebruikt zal worden is een nep hortensia welke op een vaste positie zal worden neergelegd aan het begin van elke test. Dit zal op 60cm links en 15,5cm onder het voetstuk van de arm zijn. De hortensia zal worden verplaatst tot een punt 45cm voor en op gelijke hoogste van het voetstuk van de arm. Er zal een knop worden aangesloten op de I/O pins op de control box van de Fairino. Deze knop zal een input signaal simuleren van een sensor of PLC.

De benodigdheden voor deze test zijn:

- Fairino FR 5
- Fairino Python SDK Version 2.1.6
- Python Version 3.13.2
- JODELL ERG J1.0, gemonteerd op het kopstuk van de arm en aangesloten op de output port op de arm.
- Kunststof hortensia
- Test code
- Knop

## De test

In deze test zal er een kunststof hortensia worden verplaatst doormiddel van de JODELL ERG grijper. Het proces zal gestart worden doormiddel van de gelezen input van een knop welke hardware matig is aangesloten op de control box. Na het indrukken van de knop zal de arm een hortensia oppakken en verplaatsen naar de eindlocatie. De hortensia zal op verschillende plekken worden opgepakt om te testen wat de beste plek op de hortensia om deze op te pakken.

De test zal [x aantal] keer worden uitgevoerd en de resultaten zullen in een tabel worden weergeven.

| **Test nummer** | **Grijppunt** | **Bloem gepakt** | **Bloem volledig kunnen verplaatsen** |
|---|---|---|---|
| 1 | Midden | Ja | Ja |
| 2 | Uiteinde (stam) | Ja | Nee |
| 3 | 20cm boven stam | Ja | Nee |
| 4 | Onder de kop | Ja | Ja |

## Conclusie

De bloem kan het beste opgepakt worden in het midden van de steel. Het maakt niet uit of het grijppunt precies op het midden van de steel zit of hier een klein stukje van verwijdert. Het is wel belangrijk dat de bloem wordt gepakt onder de bladeren dit door de kracht welke wordt uitgeoefend op de bloem, deze is voor de bladeren van de echte bloemen waarschijnlijk te hoog en zullen dus breken.

# Zelf ontworpen grijpers

## User story

Met deze test worden user story 2,3 & 4 uit de originele backlog getest.

## Test opstelling en benodigdheden

De testopstelling is als volgt. De Fairino FR5 is gemonteerd op een kleine pallet met een houten plank op de manier zoals beschreven is in de instructies van Fairino. Deze opstelling staat in een ruimte waarbij de FR5 vrijuit kan bewegen. De hortensia welke in de test gebruikt zal worden is een nep hortensia welke op een vaste positie zal worden neergelegd aan het begin van elke test. Dit zal op 60cm links en 15,5cm onder het voetstuk van de arm zijn. De hortensia zal worden verplaatst tot een punt 45cm voor en op gelijke hoogste van het voetstuk van de arm. Er zal een knop worden aangesloten op de I/O pins op de control box van de Fairino. Deze knop zal een input signaal simuleren van een sensor of PLC.

De benodigdheden voor deze test zijn:

- Fairino FR 5
- Fairino Python SDK Version 2.1.6
- Python Version 3.13.2
- JODELL ERG J1.0, gemonteerd op het kopstuk van de arm en aangesloten op de output port op de arm.
- Zelf ontworpen grijpers welke gemonteerd zijn op de JODELL ERG J1.0
- Kunststof hortensia
- Test code
- Knop

De gebruikte ontwerpen van de grijpers kunnen gevonden worden in de oplever set onder de map ontwerpen in documenten en onder de bijlage.

## De test

Voor deze test worden de zelf ontworpen opzet stukken voor de JODELL ERG J1.0 getest. Deze opzet stukken zullen worden getest op betrouwbaarheid en gebruiksvriendelijkheid. Om de betrouwbaarheid te testen wordt er gekeken naar de overgebleven ruimten tussen de grijpers en de bloem zelf. Om dit goed te testen zullen er meerdere bloemen met verschillende diktes worden gebruikt. Hoe kleiner de ruimte tussen de bloem en de grijper hoe betrouwbaarder de grijper is. Om de gebruiksvriendelijkheid te testen zal er worden gekeken naar de ruimte welke over is als de grijper open is. De diameter van de open grijper wordt vergeleken met de diameter van de bloemen. Ook hier zullen verschillende bloemen met verschillende diktes gebruikt worden. De groter het verschil tussen de twee diameters hoe makkelijker de bloem opgepakt kan worden aldus de hoger de gebruiksvriendelijkheid.

Alleen de ontwerpen van de grijper welke zijn geoptimaliseerd worden getest dit omdat de niet geoptimaliseerde versies niet correct sluiten of te fragiel waren voor gebruik.

| **Grijper versie** | **Overgebleven ruimte bij gesloten grijper (mm)** | **Diameter afstand opengrijper (mm)** |
|---|---|---|
| **V 0.3** | 11 mm | 19 mm |
| **V 1.3** | 14 mm | 29 mm |
| **V 2.X** | | |

# Veiligheid

In deze test worden de veiligheidsfuncties getest.

## User story

Met deze test wordt user story 7 (originele backlog) & user story 5 (controller backlog) getest.

## Test opstelling en benodigdheden

De testopstelling is als volgt. De Fairino FR5 is gemonteerd op een kleine pallet met een houten plank op de manier zoals beschreven is in de instructies van Fairino. Zorg dat deze genoeg weerstand bied om niet weg te draaien tijdens het testen. Leg desnoods extra gewicht op de pallet of verstevig deze via een andere manier. Deze opstelling staat in een ruimte waarbij de FR5 vrijuit kan bewegen.

De benodigdheden voor deze test zijn:

- Fairino FR 5
- Fairino Python SDK Version 2.1.6
- Python Version 3.13.2
- Test persoon (eigen risico)
- Test code

## Acceptatiecriteria

De test is geslaagd als voor minimaal 70% van de verplaatsingen de robotarm zal stoppen.

## De test

In deze test zal de robotarm verschillende bewegingen maken en tijdens deze beweging zal de arm worden gehinderd door een hand welke de arm tegen probeert te houden. De snelheid van de arm zal worden verhoogt per test. De druk zal komen van tegen gestelde richting. De test wordt beoordeel op het punt of de arm stopt.

| **Snelheid** | **Horizontale verplaatsing** | **Verticale verplaatsing** |
|---|---|---|
| **10%** | Ja | Ja |
| **25%** | Ja | Ja |
| **50%** | Ja | Ja |
| **75%** | Nee | Ja |
| **100%** | Nee | Nee |

Aan de hand van de test wordt duidelijk dat de arm op hogere snelheden te snel gaat om zelf te stoppen. Dit betekend dat het niet veilig is om de arm op deze snelheden te gebruiken in omgevingen waar veel mensen rond lopen. De lagere snelheden zijn wel veilig voor dit soort situaties. Ook is de hoeveelheid kracht welke nodig is om de arm te stoppen per verhoging hoger dit natuurlijk omdat de arm meer energie en kracht heeft bij hogere snelheden.

## Conclusie

De veiligheidstop van de arm is veilig genoeg om op langzame/gemiddelde snelheid te werken. Op hogere snelheid gaat de arm te snel om veilig te kunnen stoppen. Het veiligste zou zijn om de robot arm afgesloten op hoge snelheden te laten werken en een systeem te ontwikkelen wat de snelheid verlaagt wanneer er iemand in de buurt komt.

# Autonome handelingen

In deze test wordt gekeken of de software in staat is om bloemen te verplaatsen zonder menselijke input

## User story

Met deze test wordt user story 6 van de originele backlog getest.

## Test opstelling en benodigdheden

De testopstelling is als volgt. De Fairino FR5 is gemonteerd op een kleine pallet met een houten plank bovenop op de manier zoals beschreven is in de instructies van Fairino. Deze opstelling staat in een ruimte waarbij de FR5 vrijuit kan bewegen. De hortensia welke in de test gebruikt zal worden is een nep hortensia welke op een vaste positie zal worden neergelegd aan het begin van elke test. Dit zal op 60cm links en 15,5cm onder het voetstuk van de arm zijn. De hortensia zal worden verplaatst tot een punt 45cm voor en op gelijke hoogste van het voetstuk van de arm. Er zal een knop worden aangesloten op de I/O pins op de control box van de Fairino. Deze knop zal een input signaal simuleren van een sensor of PLC.

De benodigdheden voor deze test zijn:

- Fairino FR 5
- Fairino Python SDK Version 2.1.6
- Python Version 3.13.2
- JODELL ERG J1.0, gemonteerd op het kopstuk van de arm en aangesloten op de output port op de arm.
- Zelf ontworpen grijpers welke gemonteerd zijn op de JODELL ERG J1.0
- Kunststof hortensia
- Test code
- Knop

De knop wordt wel nog aangestuurd via menselijke input. Maar de functie van deze knop is om een sensor te simuleren welke geen menselijke input nodig zal hebben.

## Acceptatiecriteria

De test is geslaagd als de robot arm er 4 keer op rij in slaagt de bloem zonder problemen te kunnen verplaatsen.

## De test

In deze test zal de geschreven code worden getest of deze wel in staat is autonoom een bloem te kunnen verplaatsen. In de test zal er een sensor signaal worden gesimuleerd door een knop in te drukken. Hierna zal de robot een nep hortensia op pakken vanaf een voorbepaalde locatie en deze verplaatsen. Hierna zal de arm terug keren naar de start locatie en wachten op een nieuw signaal.

| Test nr. | Geslaagd? | Zo niet waarom niet? |
|---|---|---|
| **1** | Ja | n.v.t. |
| **2** | Ja | n.v.t. |
| **3** | Nee | Bloem niet correct opgelegd |
| **1** | Ja | n.v.t. |
| **2** | Ja | n.v.t. |
| **3** | Ja | n.v.t. |
| **4** | Ja | n.v.t. |

Aan de hand van de test wordt duidelijk dat de arm in staat is om een bloem autonoom te verplaatsen. Alleen moet deze wel goed opgelegd zijn. Een sensor welke de exacte locatie van de bloem kan doorgeven zou ideaal zijn voor deze handeling. Veder verloopt de handeling soepel en zijn er buiten de sensor om geen verbeter punten voor de handeling.

## Conclusie

De arm is instaat de bloem autonoom te verplaatsen mits deze goed is opgelegd. Een verbeter punt zou dus een sensor zijn welke de exacte locatie van de bloem door geeft aan de arm. De test is geslaagd.

# Controller input

In deze test zal de controller input getest worden op betrouwbaarheid.

## User story

Met deze test wordt user story 1 uit de controller backlog getest.

## Test opstelling en benodigdheden

Voor deze test wordt er gekeken naar de betrouwbaarheid van de pygame library. De library zal de waardes van een Dualshock 4 controller uitlezen. Deze waardes worden vergeleken met de waardes welke worden uitgelezen via een hardware tester.

De benodigdheden voor deze test zijn:

- Dualshock 4 controller
- Hardwaretester.com
- Test code

De gebruikte Dualshock 4 controller is 9 jaar oud maar heeft geen signalen van stickdrift. De resultaten kunnen variëren met een andere controller.

## Acceptatiecriteria

De test is geslaagd als de gemeten waardes tussen de library en Hardwaretester.com minder dan 0.10 zijn.

## De test

Voor deze test zullen de input waardes van de Dualshock 4 controller via de pygame library worden vergeleken met deze van Hardwaretester.com. De twee resultaten zullen worden vergeleken door de inputwaardes op behaalde punten te vergelijken met elkaar. Voor elke joystick zullen er op 5 punten vergeleken worden; meest extreme negatieve waarde, halverwege de rust positie en de meest negatieve waarde, rust positie, halverwege de rust positie en meest positieve waarde en de meest positieve waarde. Hierna zal het gemiddelde verschil worden berekent en in de tabel worden gezet. Het doel is om erachter te komen of de input waardes welke via pycharm worden verkregen betrouwbaar genoeg zijn voor gebruik.

| **Joystick** | **Gemiddelde verschil** |
|---|---|
| **As 0:** | 0.01 |
| **As 1:** | 0.02 |
| **As 2:** | 0.01 |
| **As 3:** | 0.00 |
| **As 4:** | 0.01 |
| **As 5:** | 0.03 |

De test is geslaagd de verschillen tussen de library en hardwaretester.com zijn zo minimaal dat de betrouwbaarheid van pygame gegarandeerd kan worden. Wel zal er een minimum waarde in de code gevoegd moeten worden omdat de waardes van de assen nooit compleet 0 zijn in rust.

## Conclusie

Pygame is betrouwbaar genoeg om de input waardes van de controller uit te lezen. Wel zal er een minimum waarden moeten worden ingevoegd om ervoor te zorgen dat de arm niet standaard beweegt. De test is geslaagd.

# Noodstop controller

In deze test zal de noodknop welke op de controller gebonden zal worden getest.

## User story

Met deze test wordt user story 5 van de controller backlog getest.

## Test opstelling en benodigdheden

Voor deze test wordt de noodknop functie op de dualshock 4 controller getest. De test opstelling zal als volgt zijn. De Fairino FR5 is gemonteerd op een kleine pallet met een houten plank bovenop op de manier zoals beschreven is in de instructies van Fairino. Deze opstelling staat in een ruimte waarbij de FR5 vrijuit kan bewegen. De Dualshock 4 controller zal via een bluetooth connectie worden verbonden aan de pc waar de code op zal worden gedraaid. De FR5 zal in deze test enkel een beweging maken en hoeft dus niet uitgerust te zijn met een grijper.

De benodigdheden voor deze test zijn:

- Fairino FR5
- Fairino Python SDK Version 2.1.6
- Python Version 3.13.3
- Dualshock 4 controller
- Stopwatch

## Acceptatiecriteria

De test is geslaagd als de vertraging tussen de noodknop indrukken en het stoppen van de robot onder een seconden ligt.

## De test

Voor deze test zal de FR5 een simpele beweging maken. Er zal op een willekeurig moment tijdens deze beweging op de PS knop (knop nummer invullen) worden gedrukt. Deze knop zal de beweging van de robot stop zetten en deze in de modus zetten om deze vrijuit met de hand te verzetten. De test wordt beoordeeld op de vertraging tussen het moment van de knop indrukken en het stoppen van de code.

| **Testnummer** | **Gestopt?** | **Zo ja, vertraging** |
|---|---|---|
| | | |
| | | |
| | | |

# Bijlage

Alle codes kunnen ook gevonden worden onder de map TestCodes in de opleverset

Code nep hortensia verplaatsen & Autonome handelingen

```
from fairino import Robot
import time

state = False
joint_pos = [[6.021, -111.91, -95.496, -63.832, 90, -127.14], # Pick up point + 30cm
             [-87.29,-86.502, -100.31,-83.686,90,-132.205] # Drop point + 30 cm
]
# declaratie.
tool = user = 0
vel = 100.0
flag = 0

# Safety settings
secure = [4.0,4.0,4.0,4.0,4.0,4.0]
mode = 0
config = 1

# set up main for test plan setup
robot = Robot.RPC('192.168.178.23')
def setup():


    # zet safety instellingen zodat bij contact de arm stopt
    rtn = robot.SetAnticollision(mode, secure, config)
    print(rtn)

    rtn = robot.SetCollisionStrategy(3, 2000, 100, 250)
    print(rtn)

    # Reset gripper om er zeker van te zijn dat die werkt
    # Zorg dat niks tussen de vingers zit het 0 en 100 punt wordt bepaald
    robot.ActGripper(2,0) # verander 2 naar aangesloten index
    time.sleep(1)
    robot.ActGripper(2,1)
    time.sleep(1)

def inputsensor():
    global state
    rtn, state = robot.GetDI(1) # juiste pin invullen
    if state:
        state = True
        print("Input: ", state)
        robot.SetDO(0,1)

def moveGripper(pos, force):
    rtn = robot.MoveGripper(2, pos, 100, force, 10000, 0, 0, 0, 0, 0)
    print("Moving gripper: ", rtn)
    time.sleep(1)

def move(moveTo, dif, Cart):
    if Cart == 0:
        print("Moving to")
        print(moveTo)
        rtn = robot.MoveJ(joint_pos=moveTo, tool=tool, vel=vel, user=user)
        print("MoveJ: ", rtn)
    elif Cart == 1:
        print("Changing z-axis by: ", dif)
        rtn, pos = robot.GetActualTCPPose()
```

```
        pos[2] = pos[2] + dif
        print(pos)
        time.sleep(2)
        rtn = robot.MoveCart(desc_pos=pos, vel=vel, user=user, tool=tool)

        print("MoveCart: ", rtn)
    rtn = robot.GetActualTCPPose()
    print("moved", rtn)

def main():
    moveGripper(0, 100)

    while state == False:
        inputsensor()

    # ga naar band als die daar niet al is en open grijper
    move(joint_pos[0], 0, 0)
    moveGripper(0, 100)


    # zet moveTo als 0 om de lineare bewiging te gebruiken

    # Ga omlaag en pak object en ga weer omhoog
    move(0, -290, 1)
    moveGripper(100, 100)
    move(0, 290, 1)

    # ga naar drop punt
    move(joint_pos[1], 0, 0)
    move(joint_pos[1], 0, 0)

    # ga omlaag en leg neer ga weer omhoog
    move(0, -290, 1)
    moveGripper(0, 100)
    move(0, 290, 1)

    # ga terug naar band
    move(joint_pos[0], 0, 0)
    move(joint_pos[0], 0, 0)

if __name__ == '__main__':
    setup()
    main()
    time.sleep(3)
    robot.SetDO(0,0)
    robot.ResetAllError()
    robot.CloseRPC()
robot.CloseRPC()
```

Veiligheid code

```
import time

from fairino import Robot

robot = Robot.RPC('192.168.178.23')

joint_pos = [[6.021, -111.91, -95.496, -63.832, 90, -127.14], [-87.29, -86.502, -100.31, -83.686,
90, -132.205]]

tool = 0
user = 0
vel = 100.0

level = [4.0, 4.0, 4.0, 4.0, 4.0, 4.0]
mode = 0
config = 1
speed = 25

rtn = robot.SetSpeed(speed)

rtn = robot.SetAnticollision(mode=mode, level=level, config=config)
print(rtn)

rtn = robot.SetCollisionStrategy(3, 2000, 100, 250)
print(rtn)

time.sleep(5)
rtn = robot.MoveJ(joint_pos=joint_pos[0], tool=tool, user=user, vel=vel)
print(rtn)

rtn = robot.MoveJ(joint_pos=joint_pos[1], tool=tool, user=user, vel=vel)
print(rtn)

rtn, pos = robot.GetActualTCPPose()
pos[2] += 300
rtn = robot.MoveCart(desc_pos=pos, tool=tool, user=user, vel=vel)

rtn = robot.MoveJ(joint_pos=joint_pos[0], tool=tool, user=user, vel=vel)
print(rtn)

robot.CloseRPC()
```

Code uitlezen input waardes controller

```
import pygame

pygame.init()


# This is a simple class that will help us print to the screen.
# It has nothing to do with the joysticks, just outputting the
# information.
class TextPrint:
    def __init__(self):
        self.reset()
        self.font = pygame.font.Font(None, 25)

    def tprint(self, screen, text):
        text_bitmap = self.font.render(text, True, (0, 0, 0))
        screen.blit(text_bitmap, (self.x, self.y))
        self.y += self.line_height

    def reset(self):
        self.x = 10
        self.y = 10
        self.line_height = 15

    def indent(self):
        self.x += 10

    def unindent(self):
        self.x -= 10


def main():
    # Set the width and height of the screen (width, height), and name the window.
    screen = pygame.display.set_mode((500, 700))
    pygame.display.set_caption("Joystick example")

    # Used to manage how fast the screen updates.
    clock = pygame.time.Clock()

    # Get ready to print.
    text_print = TextPrint()

    # This dict can be left as-is, since pygame will generate a
    # pygame.JOYDEVICEADDED event for every joystick connected
    # at the start of the program.
    joysticks = {}

    done = False
    while not done:
        # Event processing step.
        # Possible joystick events: JOYAXISMOTION, JOYBALLMOTION, JOYBUTTONDOWN,
        # JOYBUTTONUP, JOYHATMOTION, JOYDEVICEADDED, JOYDEVICEREMOVED
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                done = True  # Flag that we are done so we exit this loop.

            if event.type == pygame.JOYBUTTONDOWN:
                print("Joystick button pressed.")
                if event.button == 0:
                    joystick = joysticks[event.instance_id]
                    if joystick.rumble(0, 0.7, 500):
                        print(f"Rumble effect played on joystick {event.instance_id}")
```

```
            if event.type == pygame.JOYBUTTONUP:
                print("Joystick button released.")

            # Handle hotplugging
            if event.type == pygame.JOYDEVICEADDED:
                # This event will be generated when the program starts for every
                # joystick, filling up the list without needing to create them manually.
                joy = pygame.joystick.Joystick(event.device_index)
                joysticks[joy.get_instance_id()] = joy
                print(f"Joystick {joy.get_instance_id()} connencted")

            if event.type == pygame.JOYDEVICEREMOVED:
                del joysticks[event.instance_id]
                print(f"Joystick {event.instance_id} disconnected")

        # Drawing step
        # First, clear the screen to white. Don't put other drawing commands
        # above this, or they will be erased with this command.
        screen.fill((255, 255, 255))
        text_print.reset()

        # Get count of joysticks.
        joystick_count = pygame.joystick.get_count()

        text_print.tprint(screen, f"Number of joysticks: {joystick_count}")
        text_print.indent()

        # For each joystick:
        for joystick in joysticks.values():
            jid = joystick.get_instance_id()

            text_print.tprint(screen, f"Joystick {jid}")
            text_print.indent()

            # Get the name from the OS for the controller/joystick.
            name = joystick.get_name()
            text_print.tprint(screen, f"Joystick name: {name}")

            guid = joystick.get_guid()
            text_print.tprint(screen, f"GUID: {guid}")

            power_level = joystick.get_power_level()
            text_print.tprint(screen, f"Joystick's power level: {power_level}")

            # Usually axis run in pairs, up/down for one, and left/right for
            # the other. Triggers count as axes.
            axes = joystick.get_numaxes()
            text_print.tprint(screen, f"Number of axes: {axes}")
            text_print.indent()

            for i in range(axes):
                axis = joystick.get_axis(i)
                text_print.tprint(screen, f"Axis {i} value: {axis:>6.3f}")
            text_print.unindent()

            buttons = joystick.get_numbuttons()
            text_print.tprint(screen, f"Number of buttons: {buttons}")
            text_print.indent()

            for i in range(buttons):
                button = joystick.get_button(i)
                text_print.tprint(screen, f"Button {i:>2} value: {button}")
            text_print.unindent()
```

```
            hats = joystick.get_numhats()
            text_print.tprint(screen, f"Number of hats: {hats}")
            text_print.indent()

            # Hat position. All or nothing for direction, not a float like
            # get_axis(). Position is a tuple of int values (x, y).
            for i in range(hats):
                hat = joystick.get_hat(i)
                text_print.tprint(screen, f"Hat {i} value: {str(hat)}")
            text_print.unindent()

            text_print.unindent()

        # Go ahead and update the screen with what we've drawn.
        pygame.display.flip()

        # Limit to 30 frames per second.
        clock.tick(30)


if __name__ == "__main__":
    main()
    # If you forget this line, the program will 'hang'
    # on exit if running from IDLE.
    pygame.quit()
```