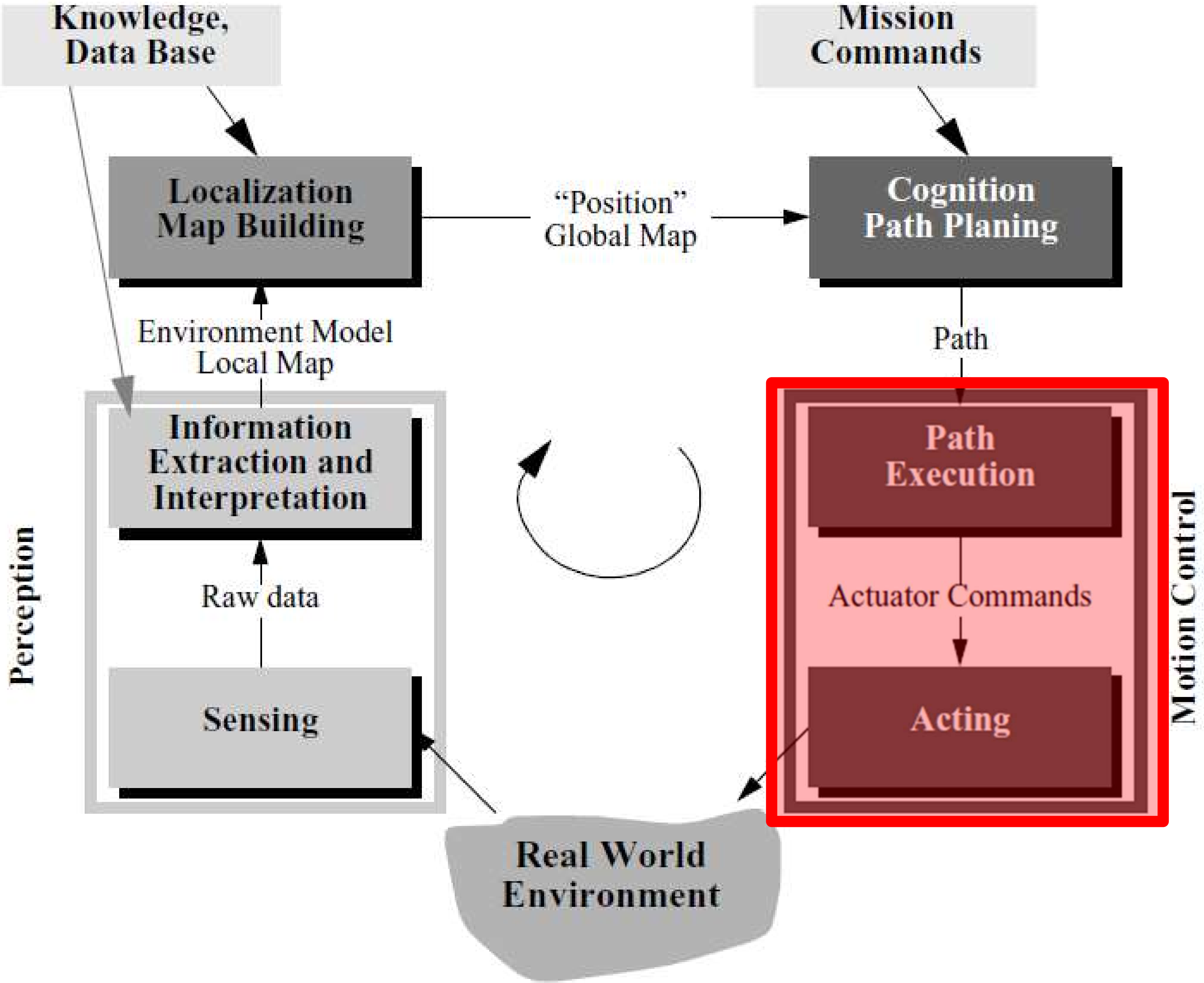
ENPM 809T

UMCP, Mitchell


Knowledge,
Data Base
Mission
Commands
Localization
Map Building
"Position"
Global Map
Cognition
Path Planing
Environment Model
Local Map
Path
Information
Extraction and
Interpretation
Path
Execution
Perception
Raw data
Actuator Commands
Motion Control
Sensing
Acting
Real World
Environment

Assembly

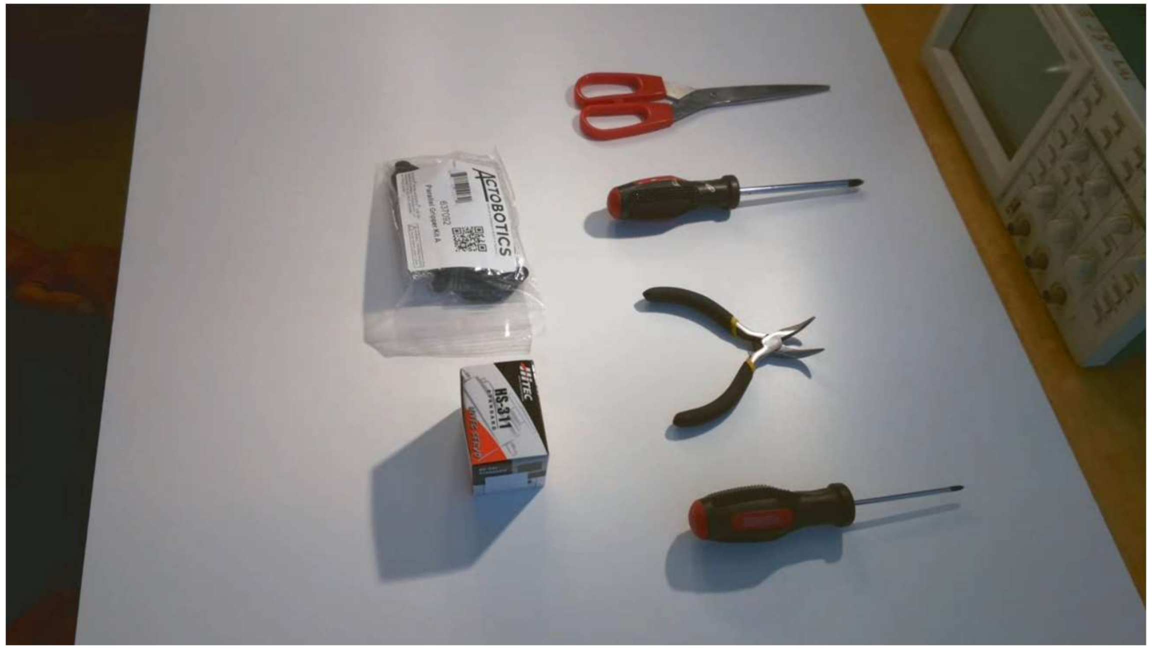
- Materials required to assemble servo motor:

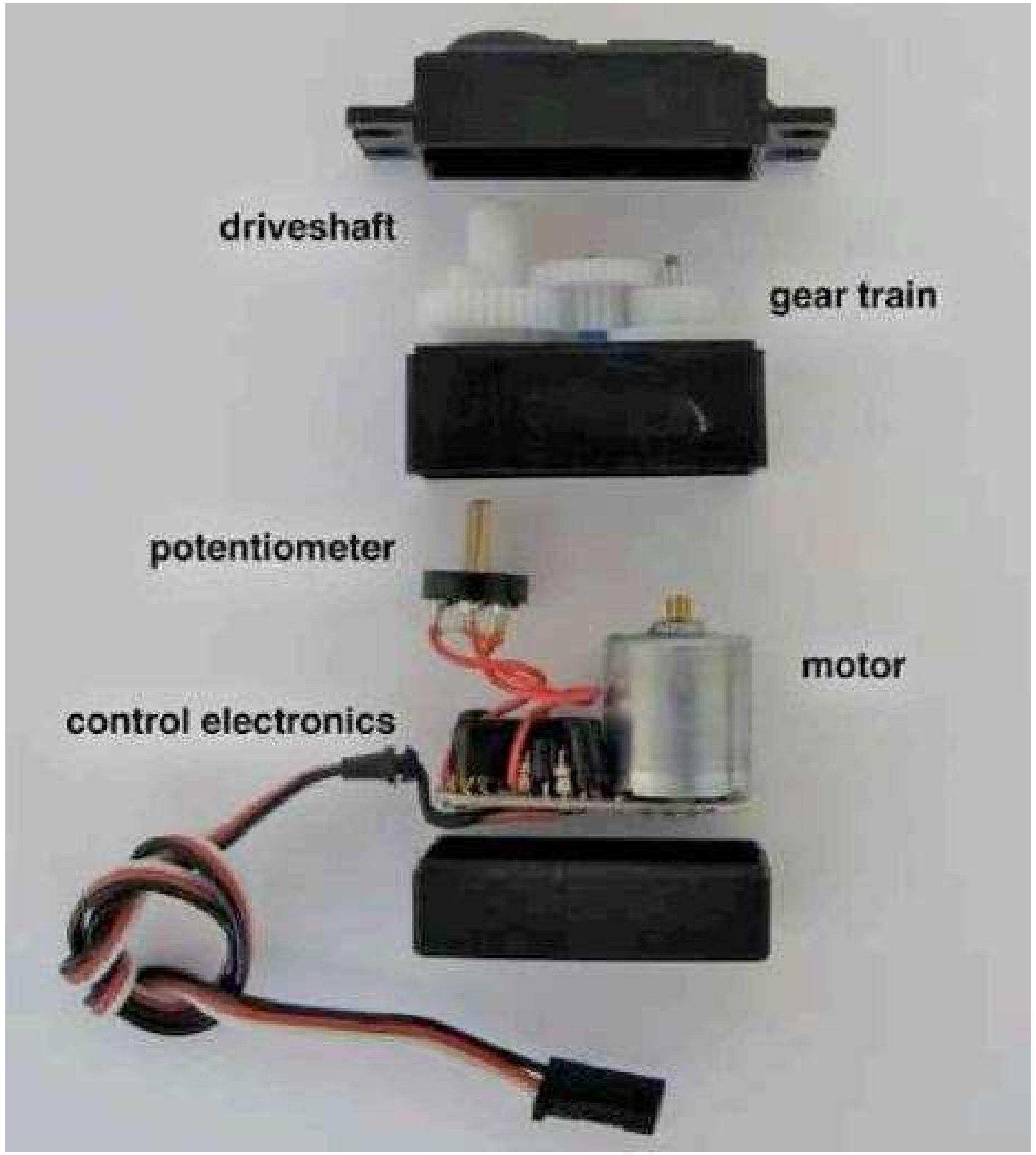
1. Servo motor
2. Gripper kit
3. Screwdrivers
4. Pliers
5. Miscellaneous wires

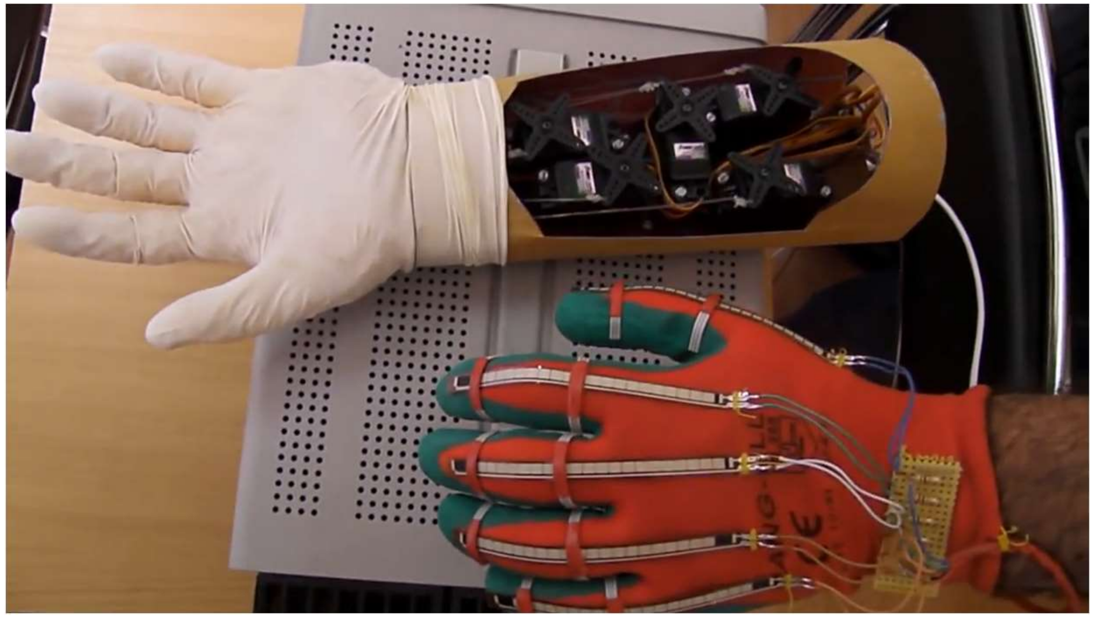
https://www.youtube.com/watch?v=poCsai98FII

Hitec servo

- Unpack materials

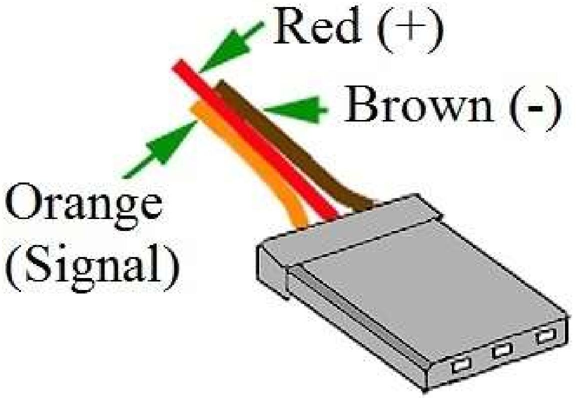



driveshaft
gear train
potentiometer
motor
control electronics

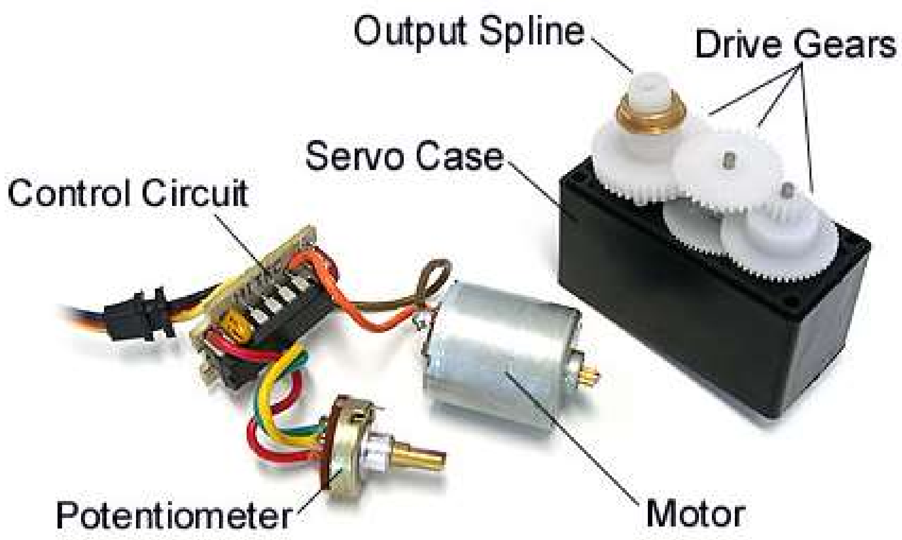


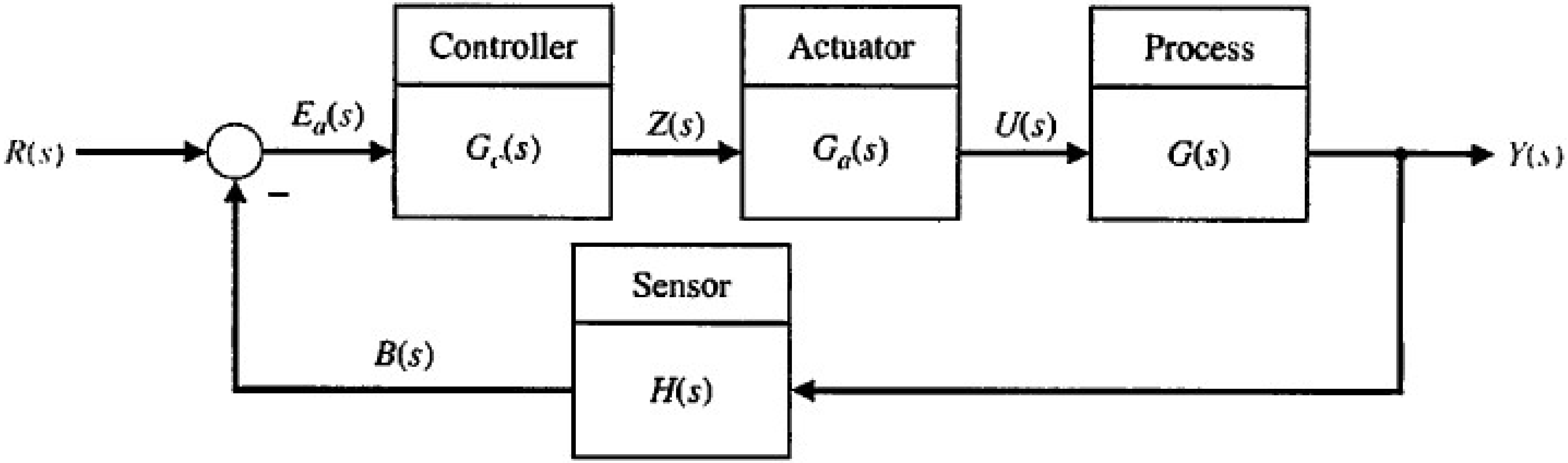
TOWER PRO
Micro Servo 9g

OSOYOO™
Micro Servo Motor
SG90


Red (+)
Brown (-)
Orange (Signal)

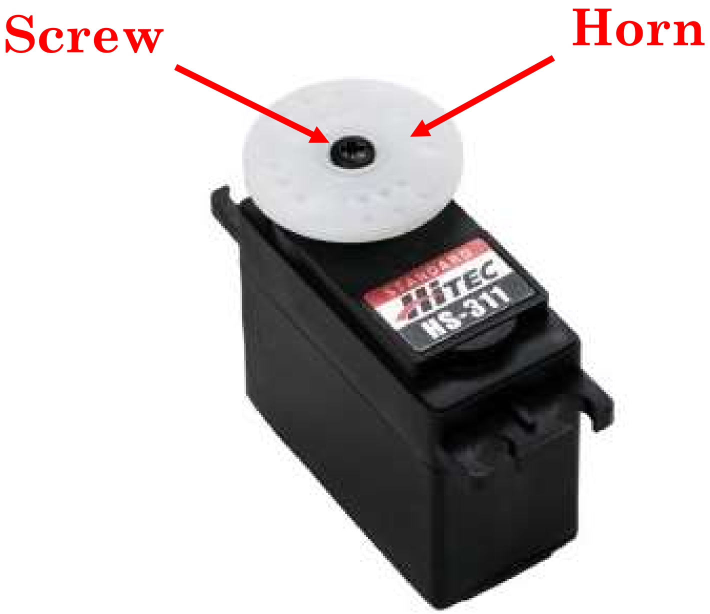
Control Theory





Hitec servo

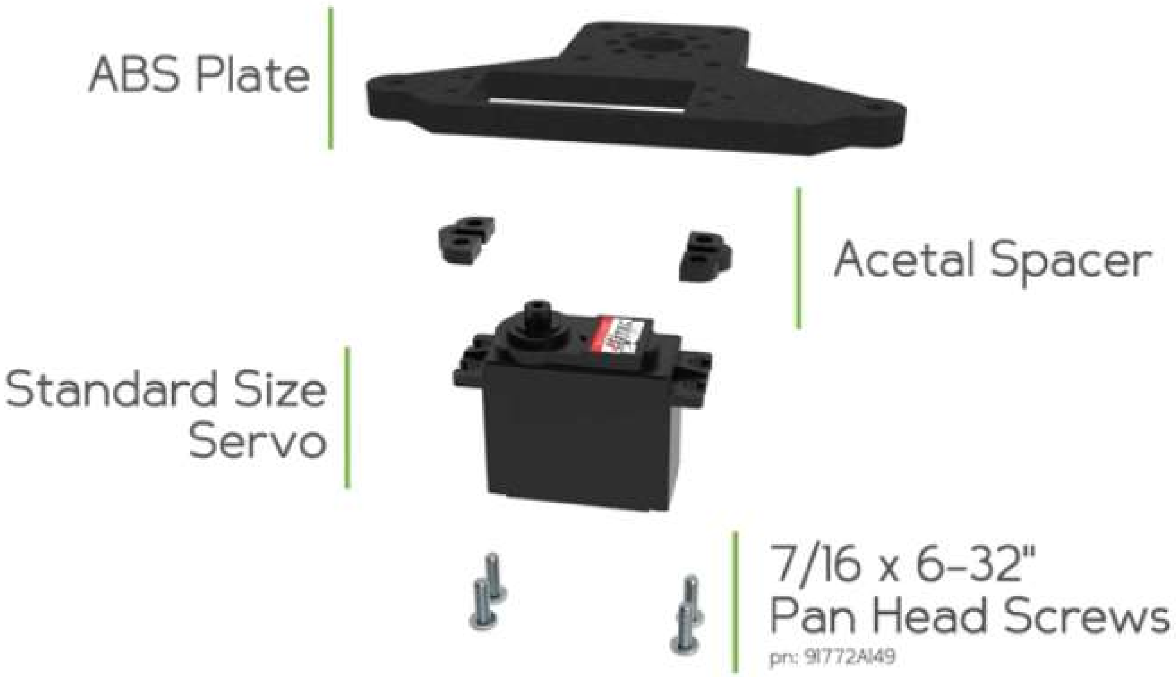
- Remove horn from servo
- Store horn in servo box
- Keep horn screw for assembly



Assembly

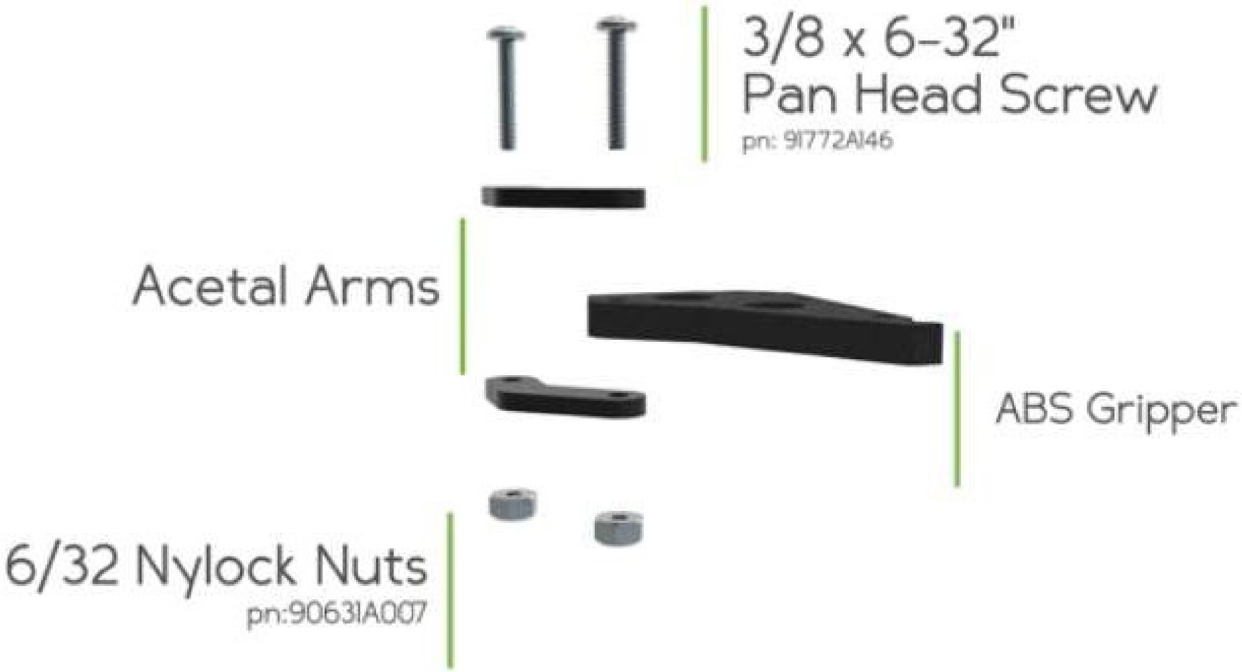
- Assemble servo to main plate

Leave all screws 1-2 turns loose for now



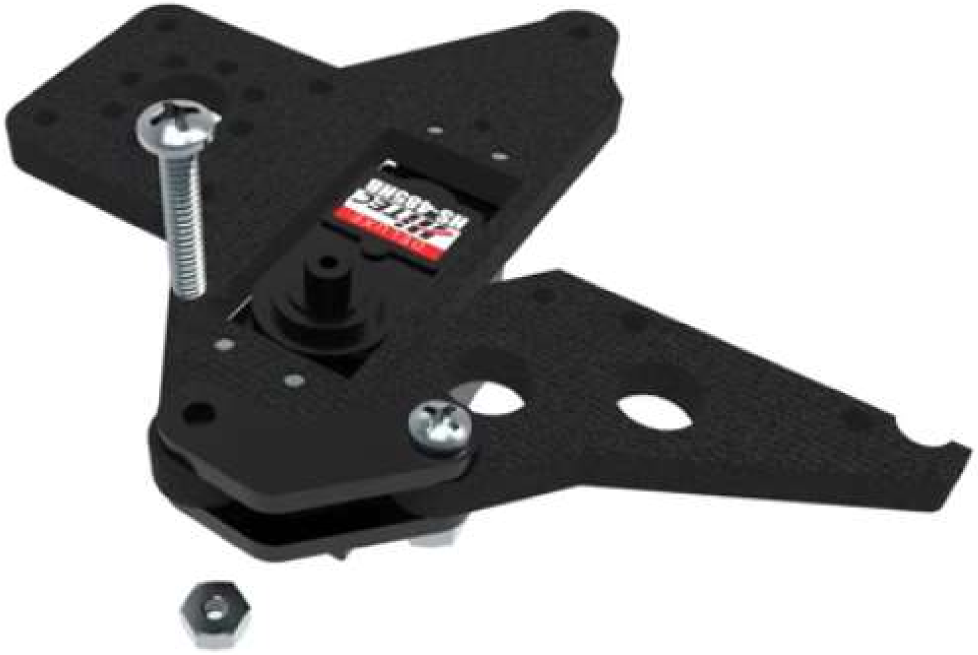
Assembly

- Assemble 1st gripper arm



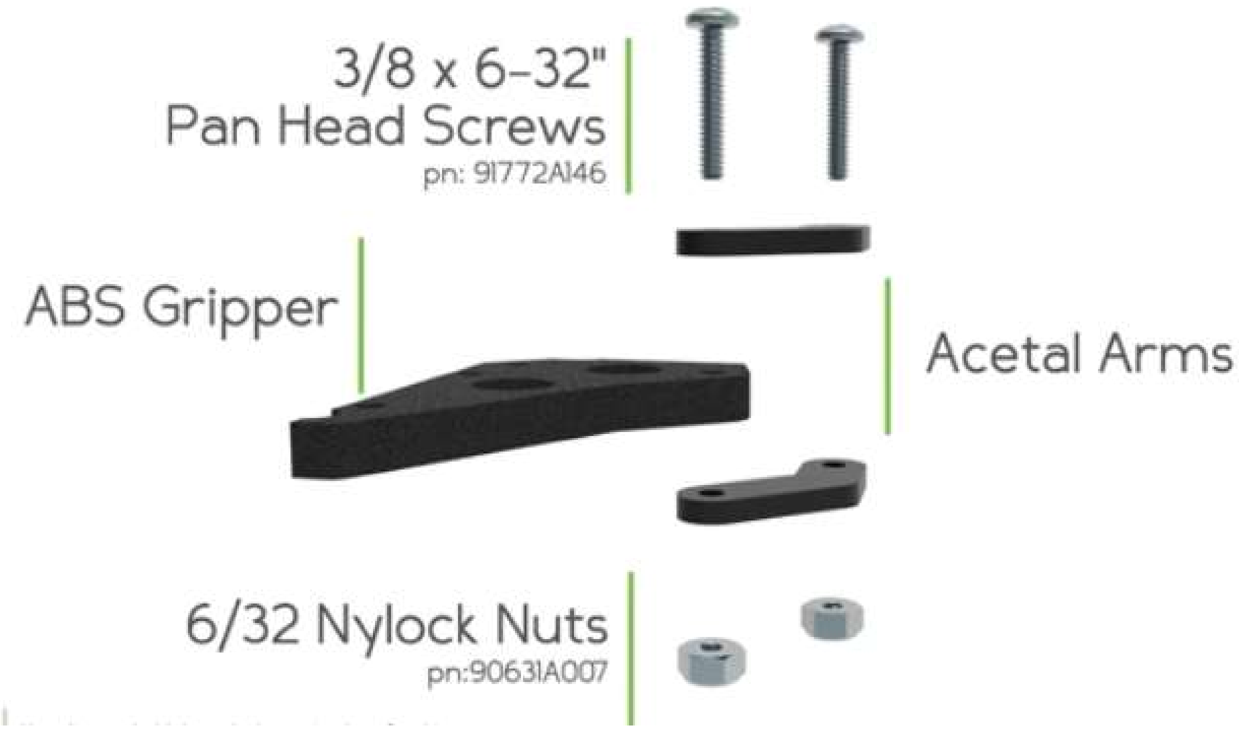
Assembly

- Mount 1st gripper arm



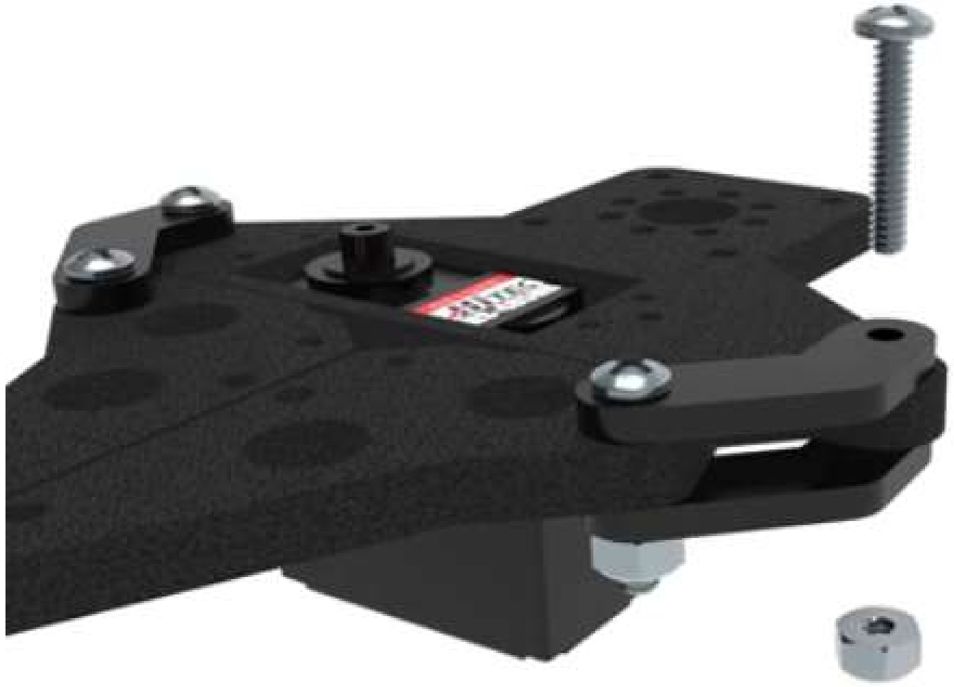
Assembly

- Assemble 2nd gripper arm



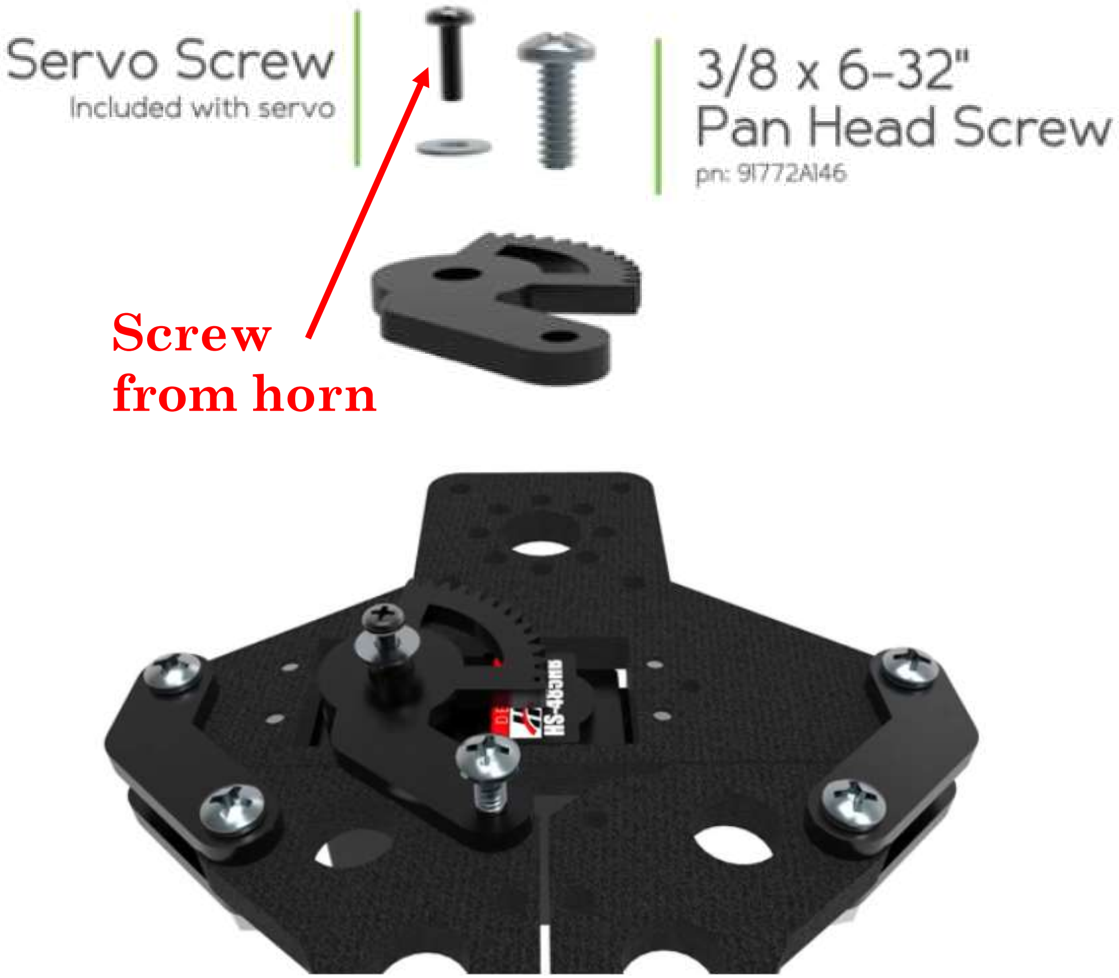
Assembly

- Mount 2nd gripper arm



Assembly

- Mount 1st geared arm

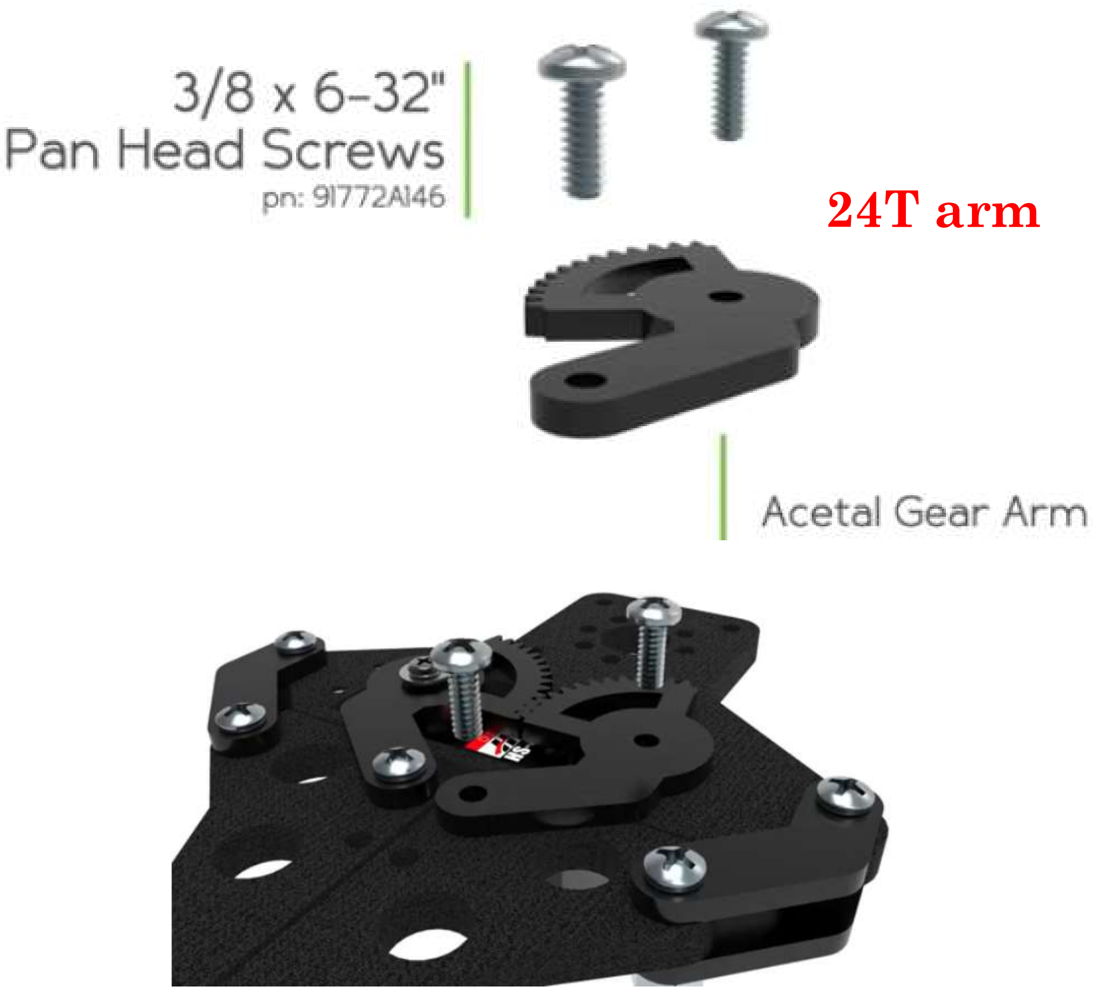


Screw from horn

May require indexing servo motor …confirm full range of servo motion

Assembly

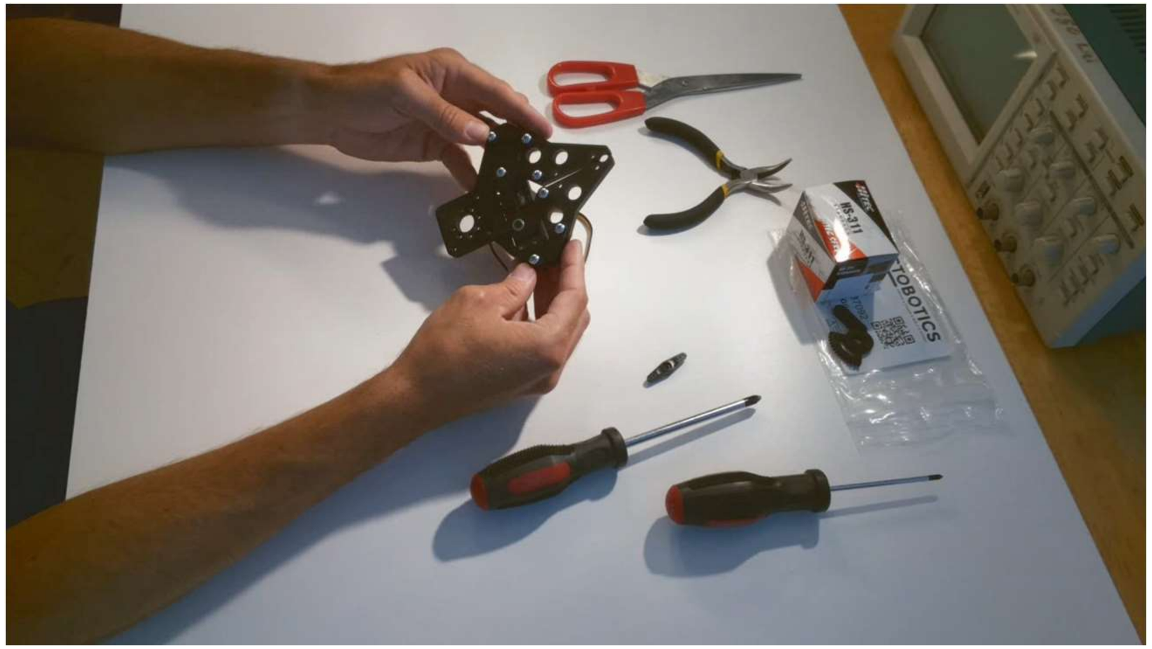
- Mount 2^{nd} geared arm



24T arm

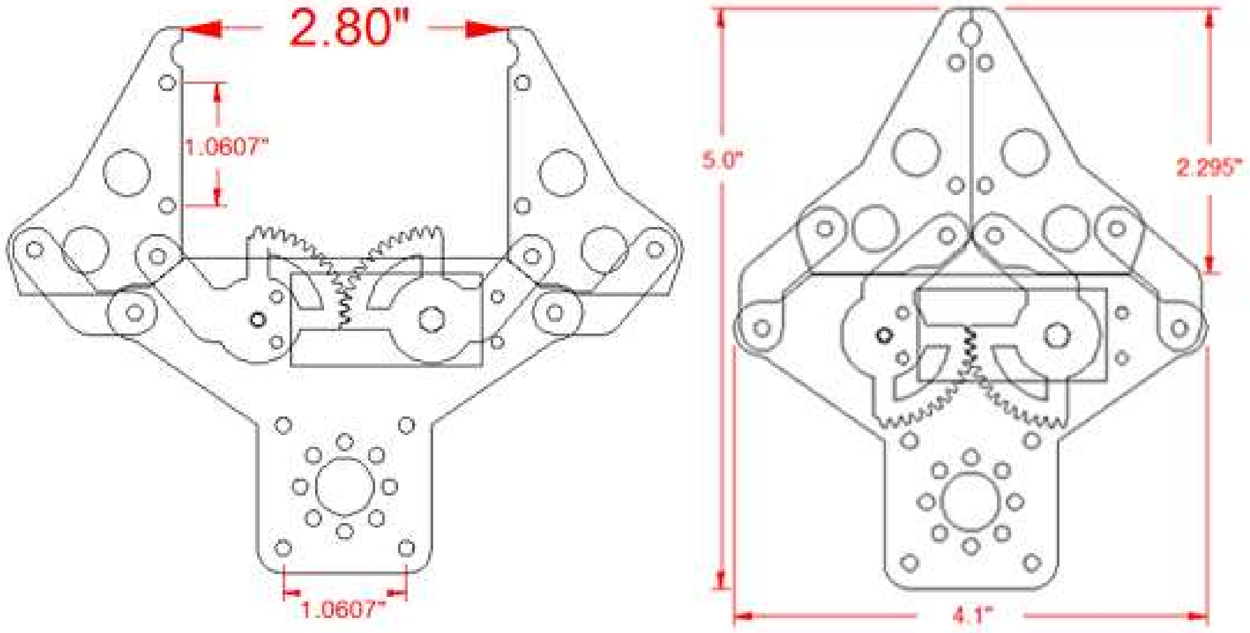
Assembly

- Secure all remaining fasteners
- When fastening rotating components: tighten each screw, then **unscrew ¼ - ½ of a turn** to permit rotation
- **Verify** gripper functions after securing each fastener



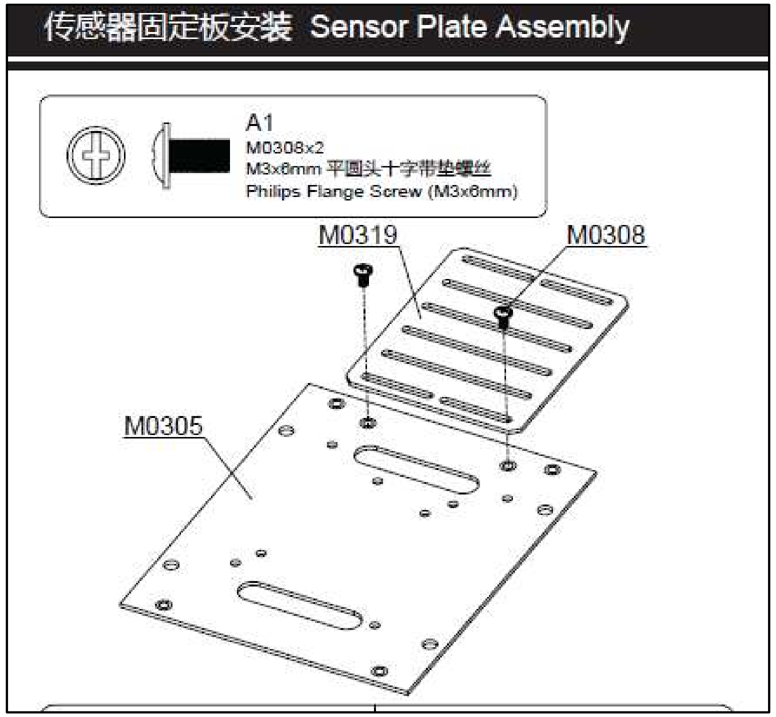
Assembly

- Confirm proper assembly
- Confirm both arms move freely

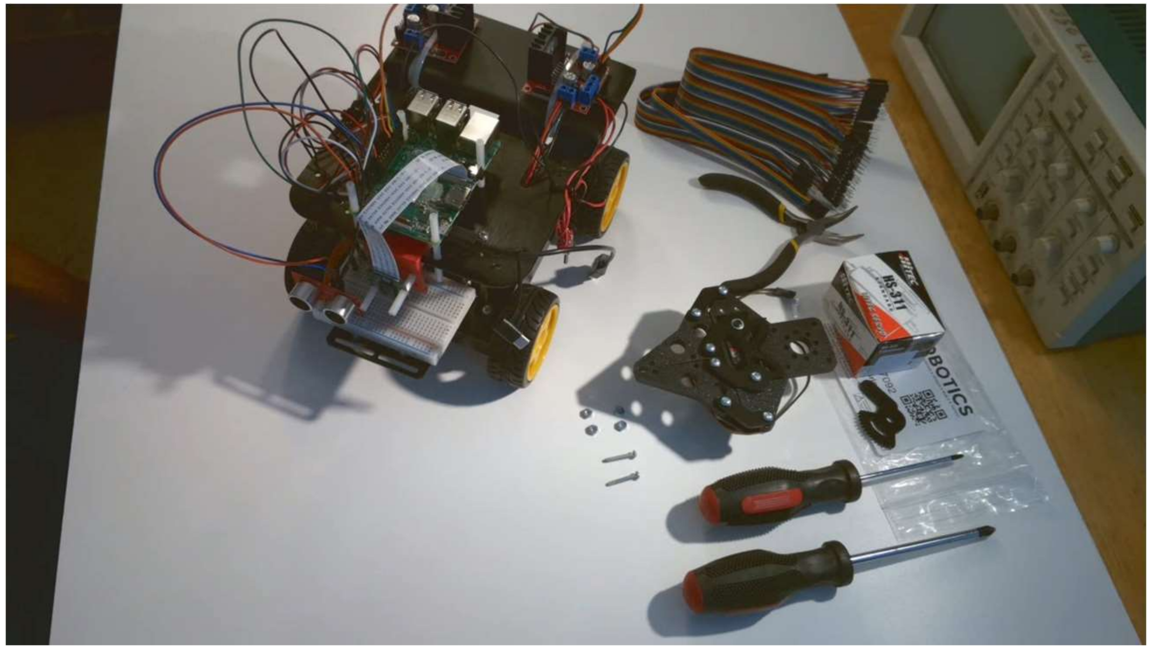


Assembly

- Mount gripper arm onto sensor plate using 2x #4-40 screws
- Fasten **each screw with 2x nuts**

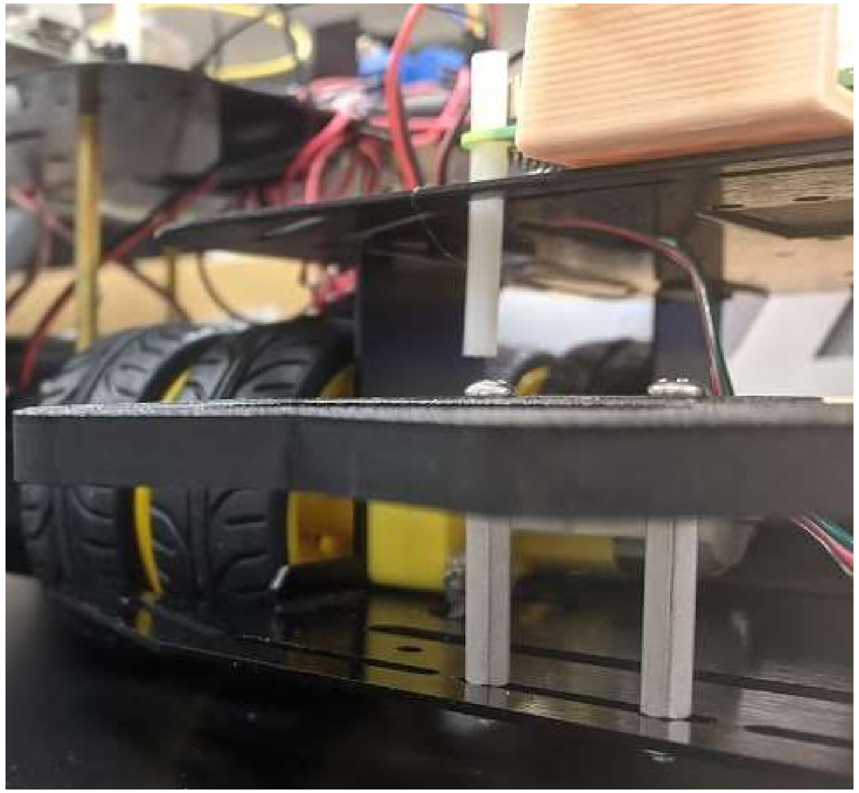

传感器固定板安装 Sensor Plate Assembly
A1
M0308x2
M3x6mm 平圆头十字带垫螺丝
Philips Flange Screw (M3x6mm)
M0319
M0308
M0305

Pirate

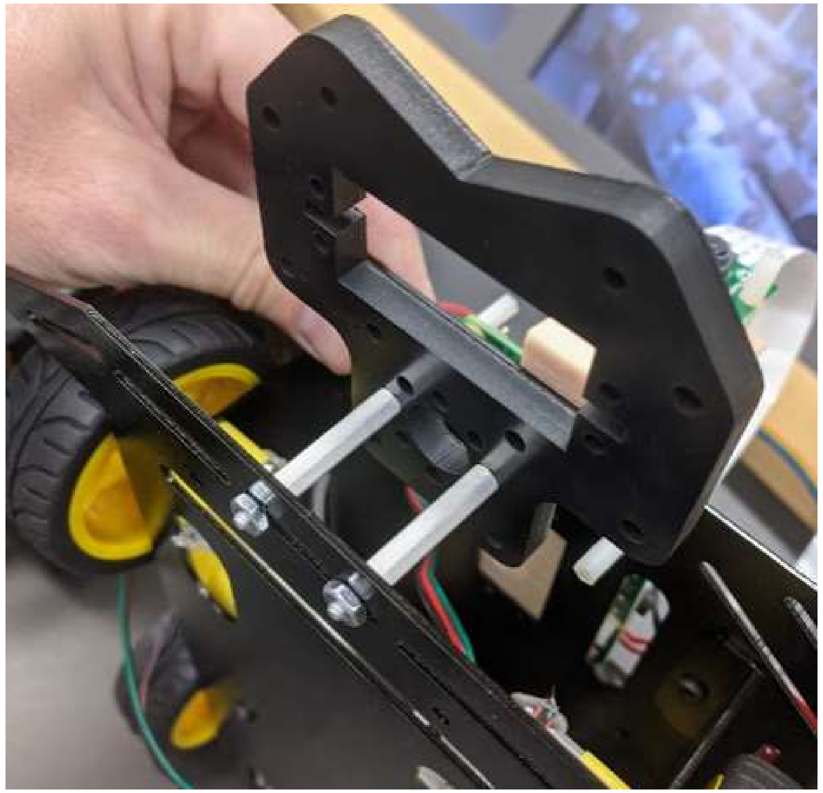


Assembly

- Mount gripper arm onto sensor plate using screws & standoffs
- Fasten **each standoff with 2x nuts**

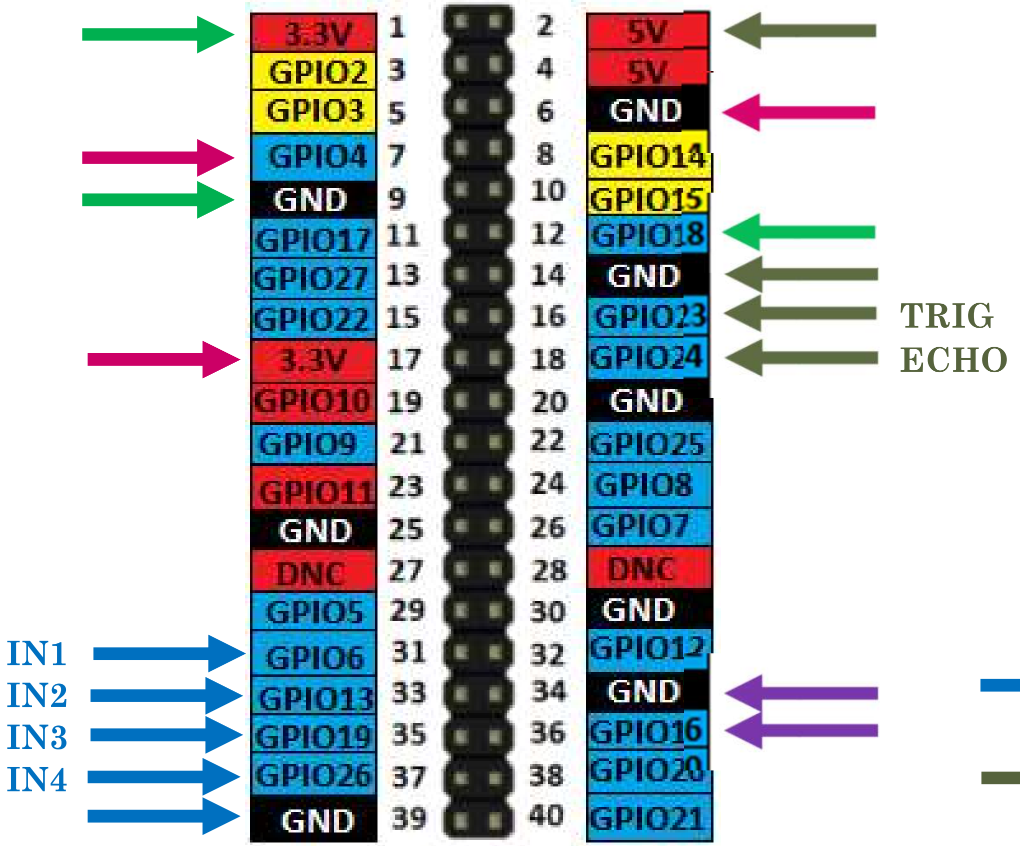


Baron

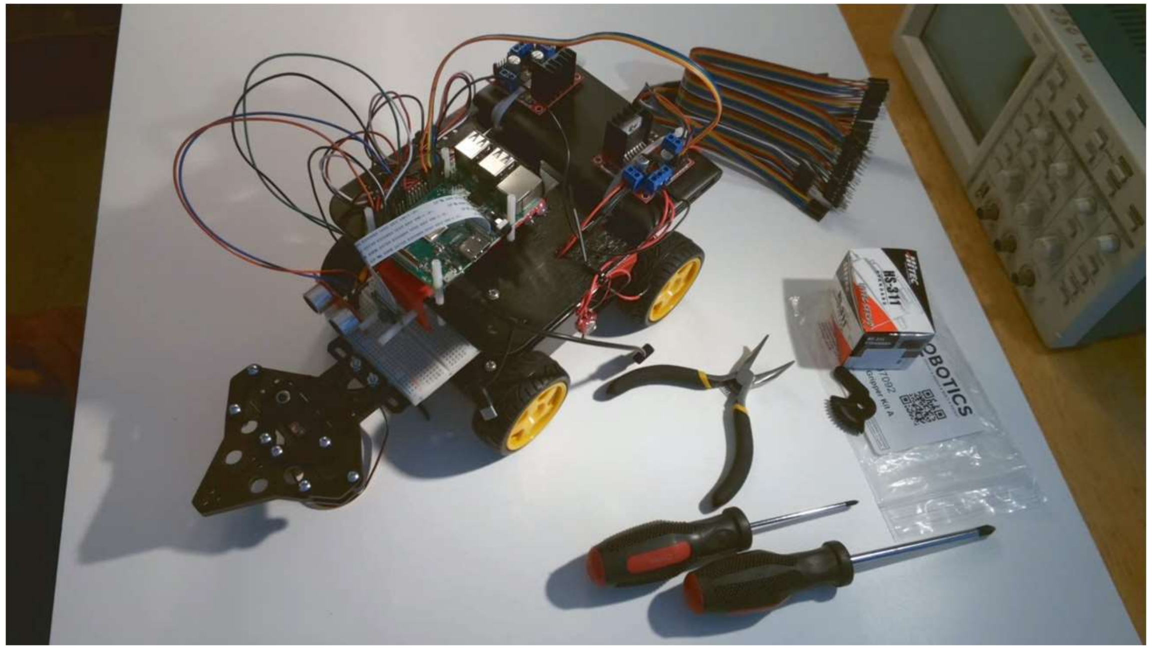


Assembly

- Wire servo motor to Raspberry Pi



3.3V	1	2	5V
GPIO2	3	4	5V
GPIO3	5	6	GND
GPIO4	7	8	GPIO14
GND	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10	19	20	GND
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
GND	25	26	GPIO7
DNC	27	28	DNC
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21



RPi to H-bridge

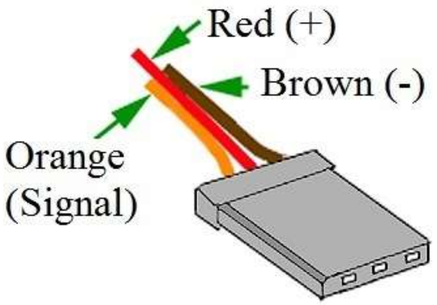
Distance sensor

Servo Motor

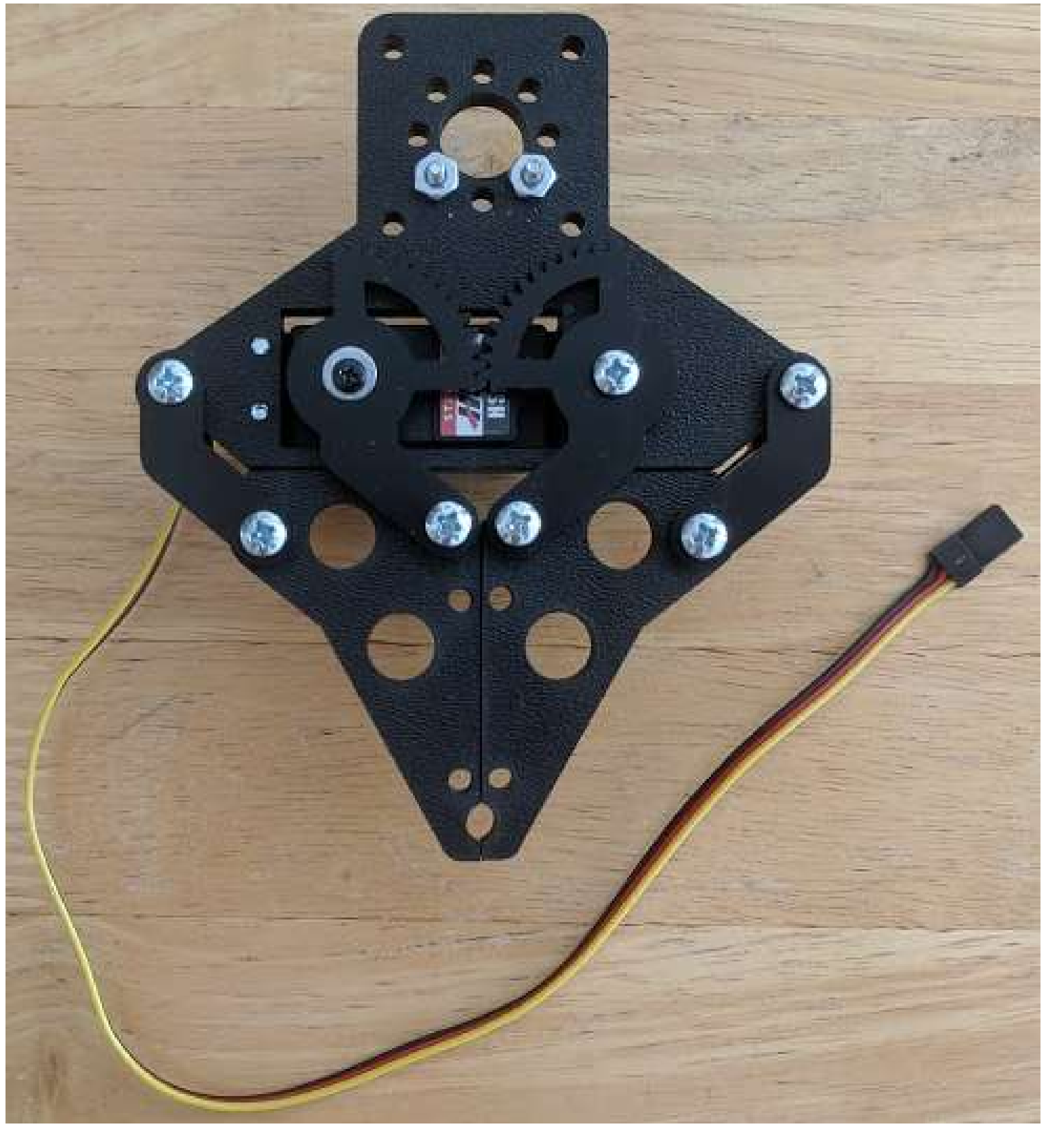
Right encoder

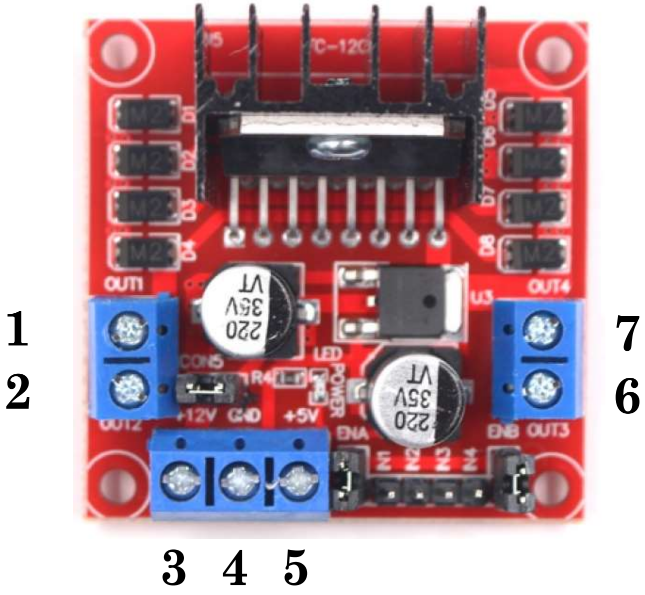
Left encoder

Servo Motor Circuit



- Raspberry Pi (should not) cannot power servo directly!
- Here we **use the AA battery pack to power servo**, through the H-bridge
- Plug **red** male-male wire into H-bridge pin 5
  - **5V** motor supply
- Plug **black** male-female wire into RPi pin 34
  - **GND**
- Plug yellow male-female wire into RPi pin 36
  - Motor control



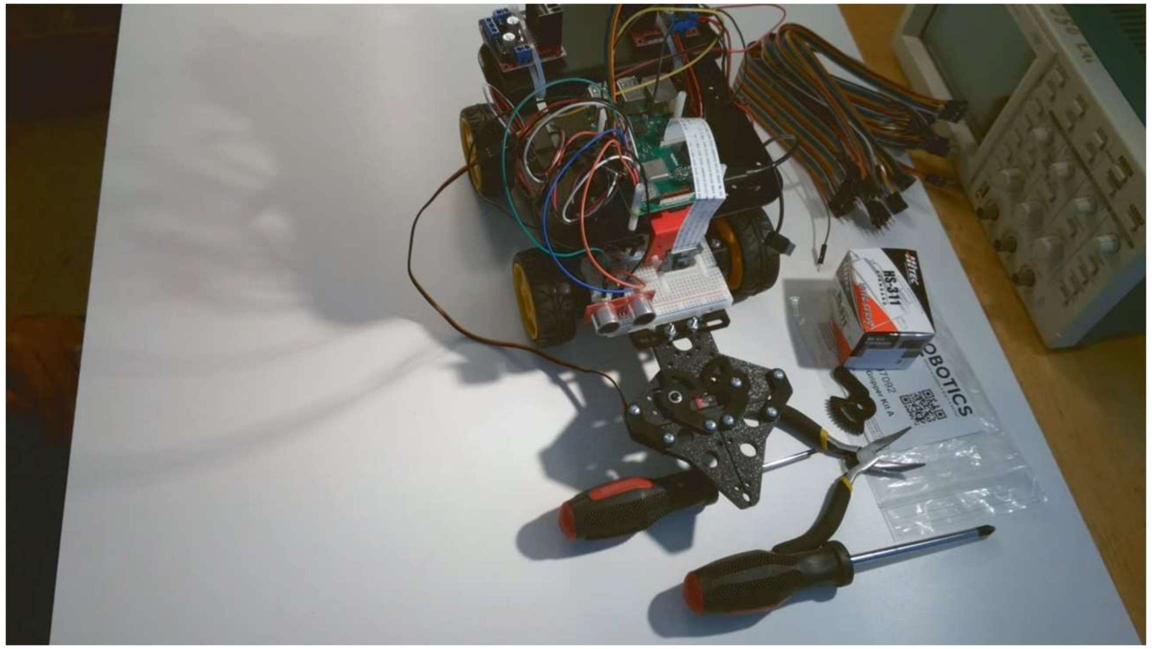


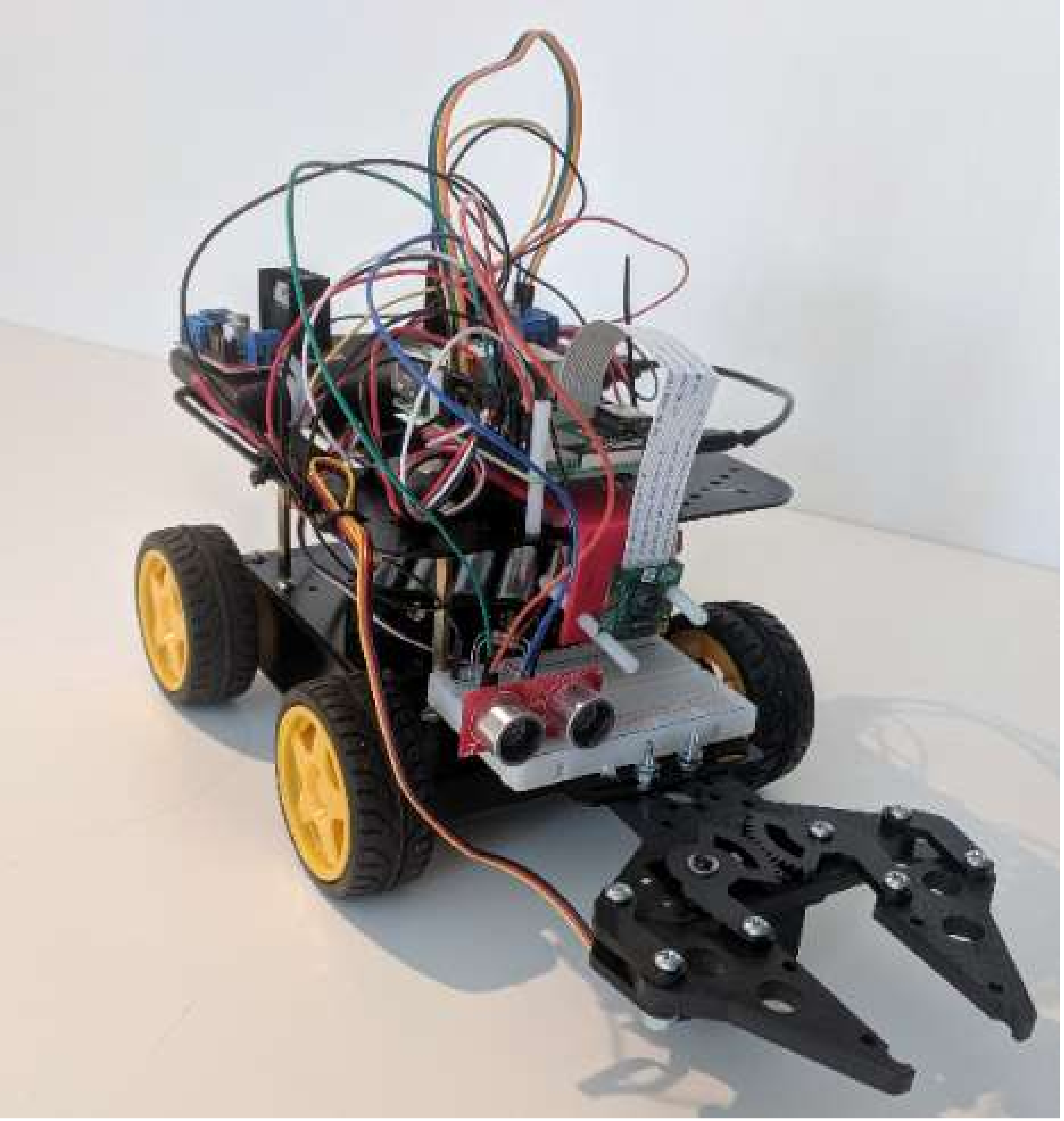
1: “+” voltage, left motors
2: “-” voltage, left motors
3: **power in** from AA battery pack
4: **GND** from AA battery pack & Raspberry Pi

5: 5V output to power servo
6: “+” voltage, right motors
7: “-” voltage, right motors

Assembly

- Secure servo motor wires with zipties

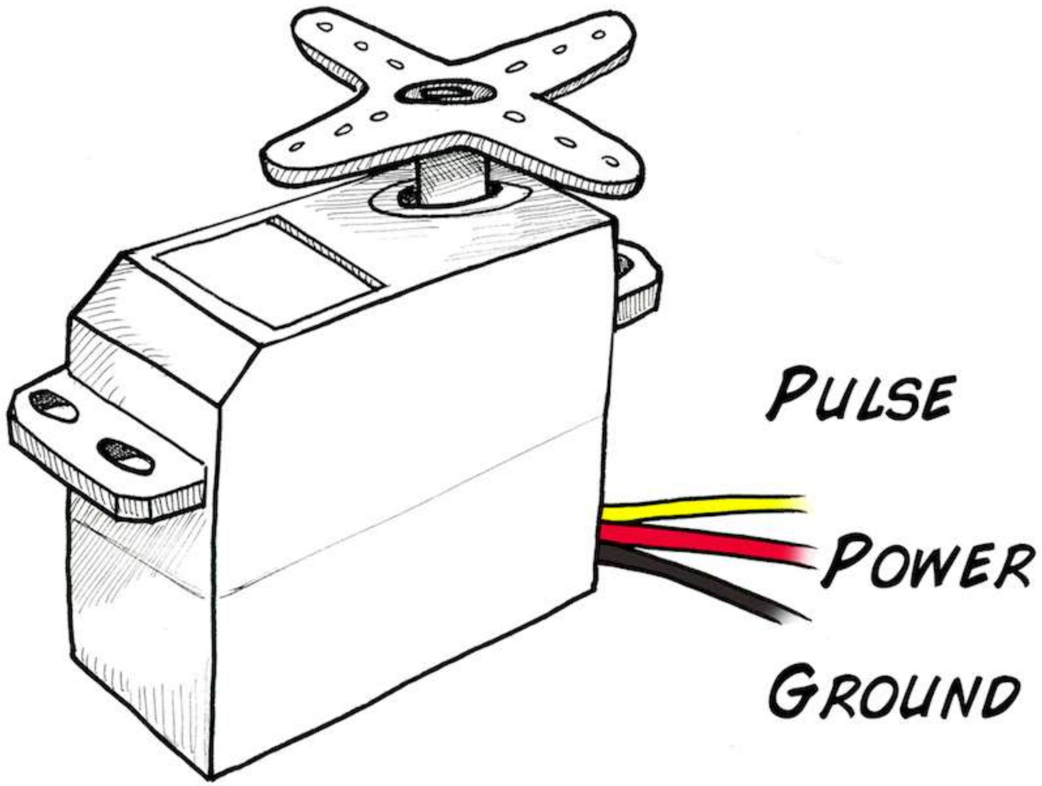


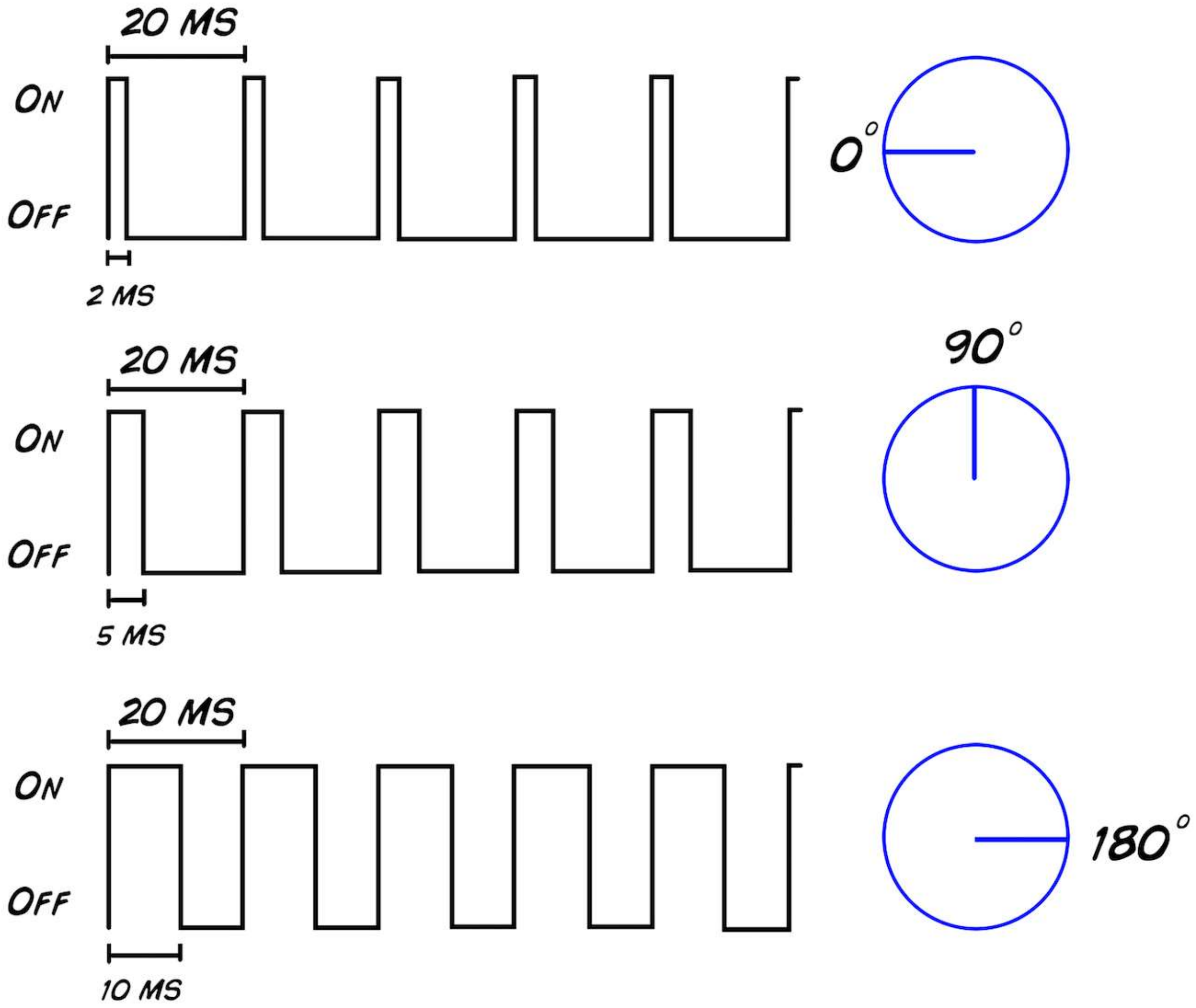


Assembly

- **Double check all electrical connections!**
- Power Raspberry Pi via USB battery pack
- Connect to Raspberry Pi using Putty/Terminal and VNC

Timing signals





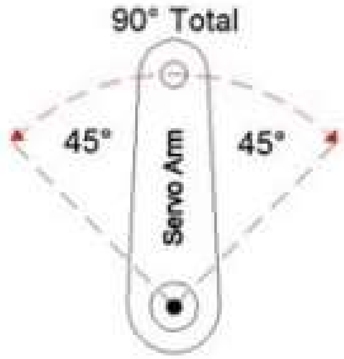
https://backyardbrains.com/experiments/MuscleSpikerShield_GripperHand

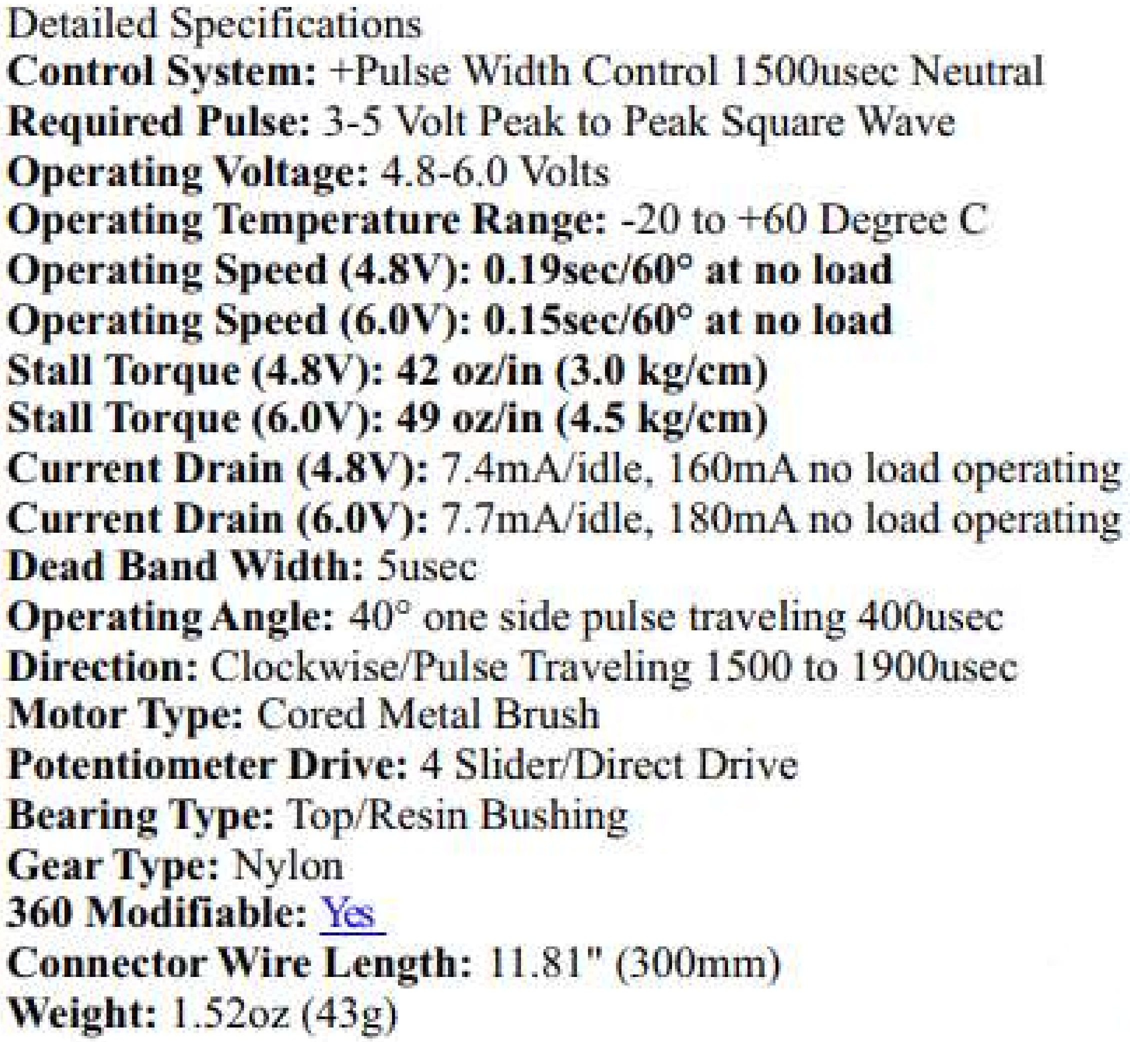
* Disclaimer *

- It is **vitally important** that the timing signal bounds are not violated
- In particular, the upper bound
- In general, failure of a servo motor is a one-time, unfixable occurrence

Timing signals

- Most servo motors use a 50 Hz control signal
- Position of servo is a function of pulse width of “On” part of cycle
- Most servos:
  - Full left ~1 msec
  - Center ~1.5 msec
  - Full right ~2 msec





Detailed Specifications

Control System: +Pulse Width Control 1500usec Neutral

Required Pulse: 3-5 Volt Peak to Peak Square Wave

Operating Voltage: 4.8-6.0 Volts

Operating Temperature Range: -20 to +60 Degree C

Operating Speed (4.8V): 0.19sec/60° at no load

Operating Speed (6.0V): 0.15sec/60° at no load

Stall Torque (4.8V): 42 oz/in (3.0 kg/cm)

Stall Torque (6.0V): 49 oz/in (4.5 kg/cm)

Current Drain (4.8V): 7.4mA/idle, 160mA no load operating

Current Drain (6.0V): 7.7mA/idle, 180mA no load operating

Dead Band Width: 5usec

Operating Angle: 40° one side pulse traveling 400usec

Direction: Clockwise/Pulse Traveling 1500 to 1900usec

Motor Type: Cored Metal Brush

Potentiometer Drive: 4 Slider/Direct Drive

Bearing Type: Top/Resin Bushing

Gear Type: Nylon

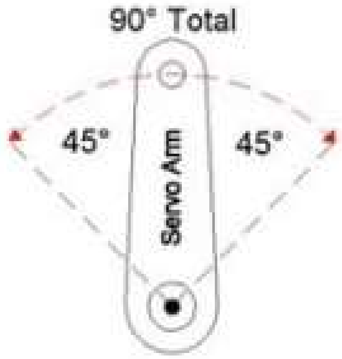
360 Modifiable: Yes

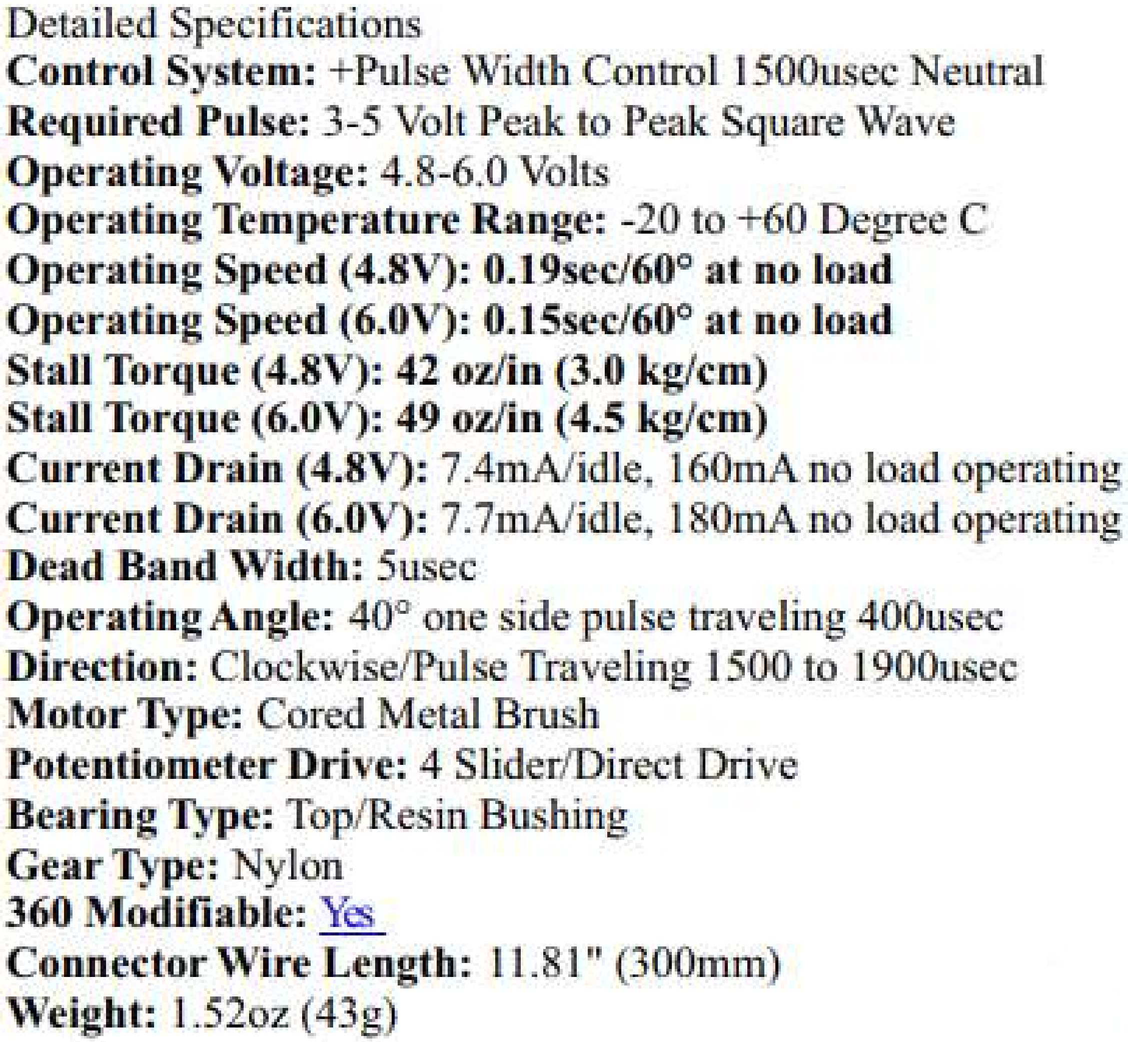
Connector Wire Length: 11.81" (300mm)

Weight: 1.52oz (43g)

Timing signals

- The period T = 1/f
- At 50 Hz, T = 1/50 = 20 msec
- Full left:
  - 1 msec = duty cycle of **5%**
  - 0.05 x 20 msec = 1 msec
- Center:
  - 1.5 msec = duty cycle of **7.5%**
  - 0.075 x 20 msec = 1.5 msec
- Full right:
  - 2 msec = duty cycle of **10%**
  - 0.1 x 20 msec = 2 msec




Detailed Specifications
Control System: +Pulse Width Control 1500usec Neutral
Required Pulse: 3-5 Volt Peak to Peak Square Wave
Operating Voltage: 4.8-6.0 Volts
Operating Temperature Range: -20 to +60 Degree C
Operating Speed (4.8V): 0.19sec/60° at no load
Operating Speed (6.0V): 0.15sec/60° at no load
Stall Torque (4.8V): 42 oz/in (3.0 kg/cm)
Stall Torque (6.0V): 49 oz/in (4.5 kg/cm)
Current Drain (4.8V): 7.4mA/idle, 160mA no load operating
Current Drain (6.0V): 7.7mA/idle, 180mA no load operating
Dead Band Width: 5usec
Operating Angle: 40° one side pulse traveling 400usec
Direction: Clockwise/Pulse Traveling 1500 to 1900usec
Motor Type: Cored Metal Brush
Potentiometer Drive: 4 Slider/Direct Drive
Bearing Type: Top/Resin Bushing
Gear Type: Nylon
360 Modifiable: Yes
Connector Wire Length: 11.81" (300mm)
Weight: 1.52oz (43g)

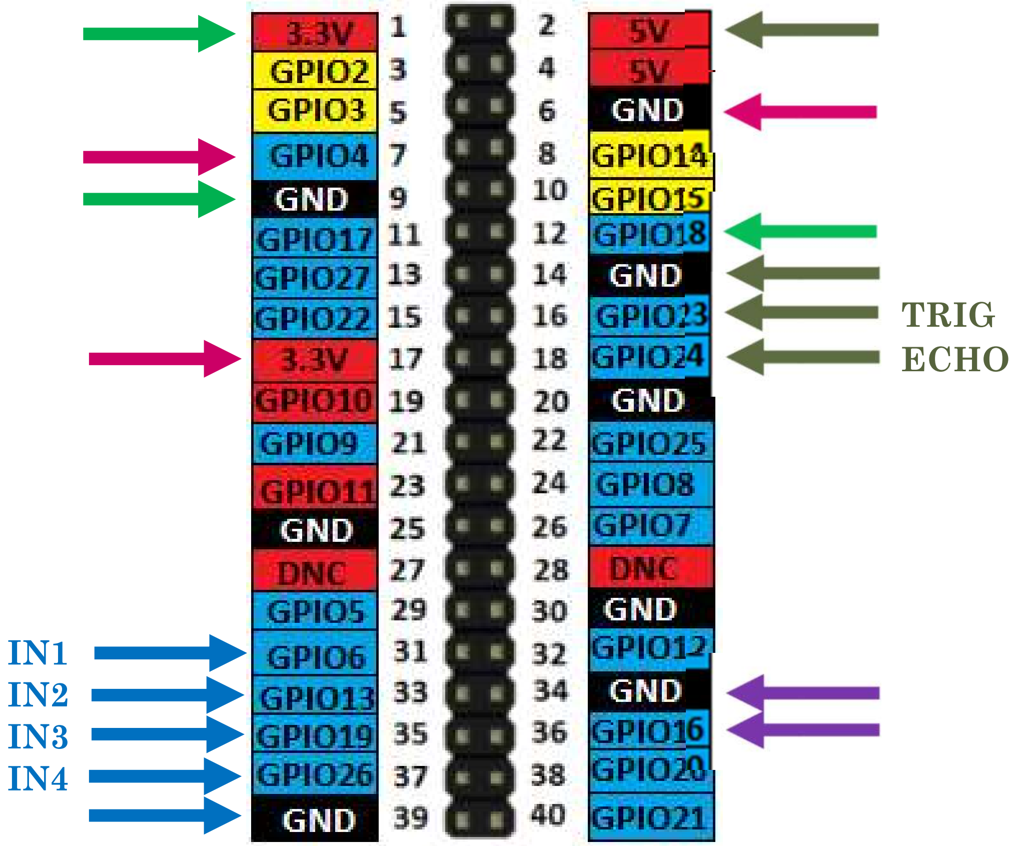
Timing signals

- The period T = 1/f
- At 50 Hz, T = 1/50 = 20 msec
- Full left:
  - 1 msec = duty cycle of **5%**
  - 0.05 x 20 msec = 1 msec
- Center:
  - 1.5 msec = duty cycle of **7.5%**
  - 0.075 x 20 msec = 1.5 msec
- Full right:
  - 2 msec = duty cycle of **10%**
  - 0.1 x 20 msec = 2 msec

VITALLY IMPORTANT NOT TO VIOLATE THIS UPPER LIMIT!!

Teleoperation

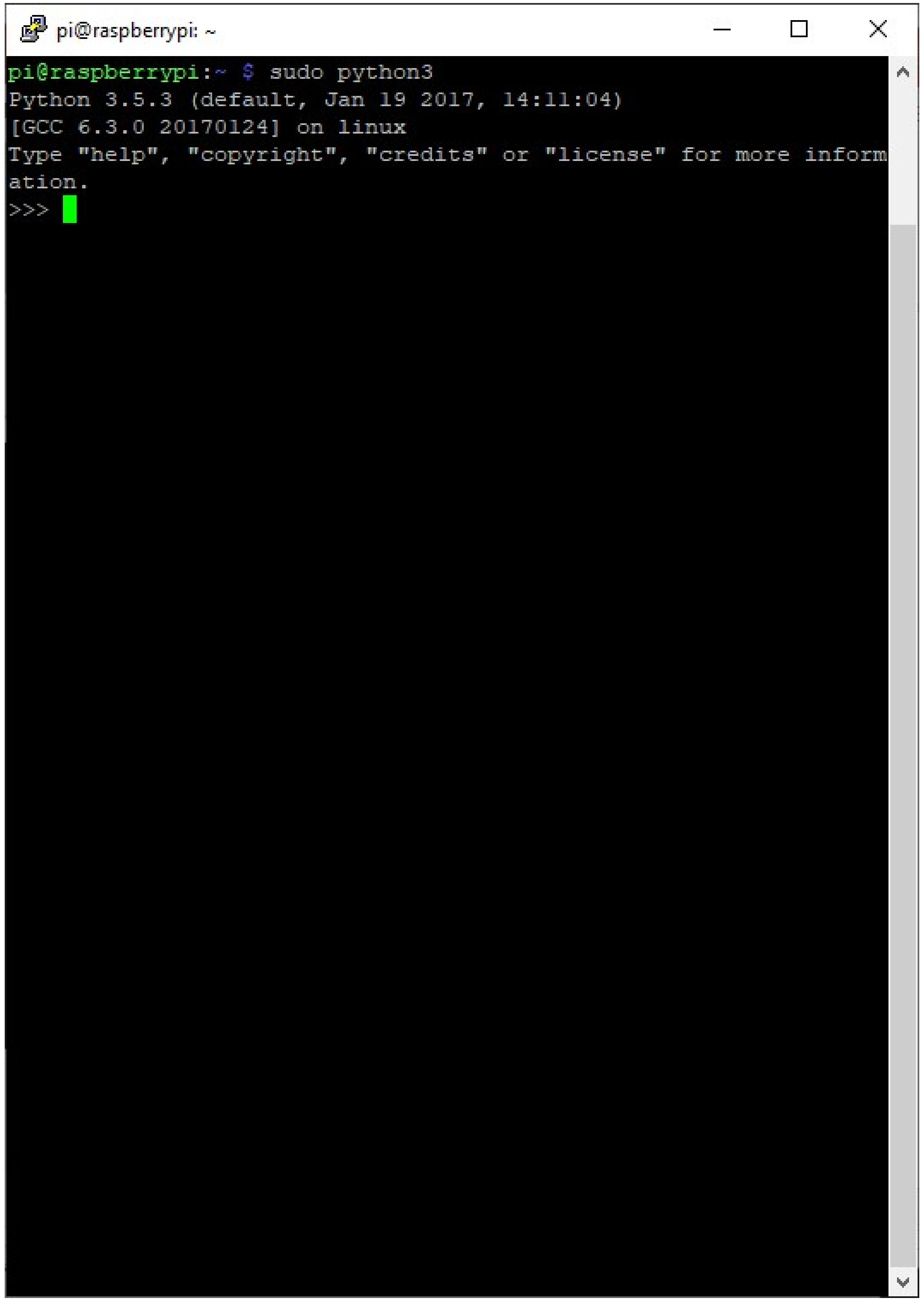
- RPi.GPIO library
- Utilize PWM functionality



https://pypi.org/project/RPi.GPIO/

Teleoperation

- Enter python3 shell

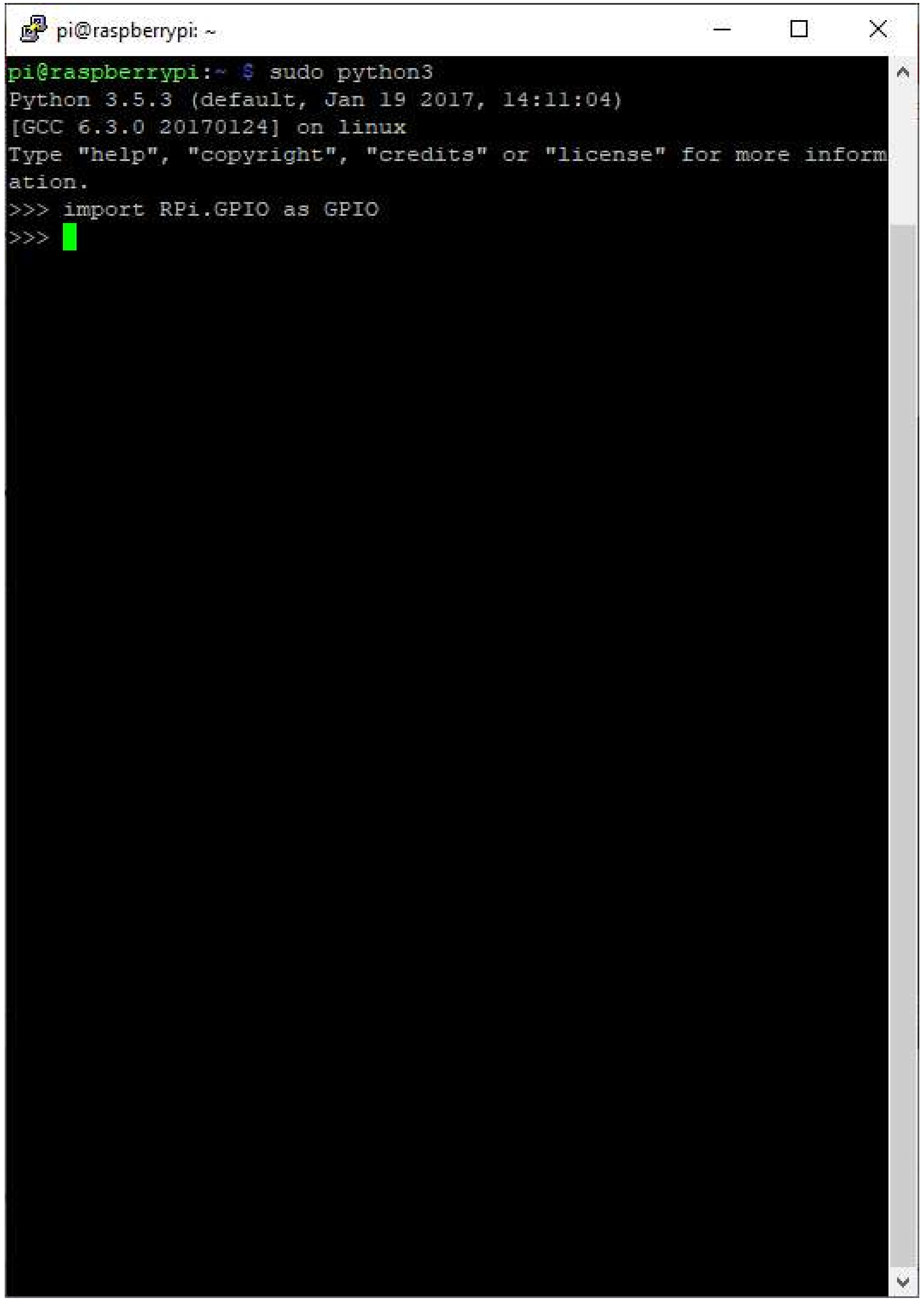


pi@raspberrypi: ~

```
pi@raspberrypi:~ $ sudo python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more inform
ation.
>>> 
```

Teleoperation

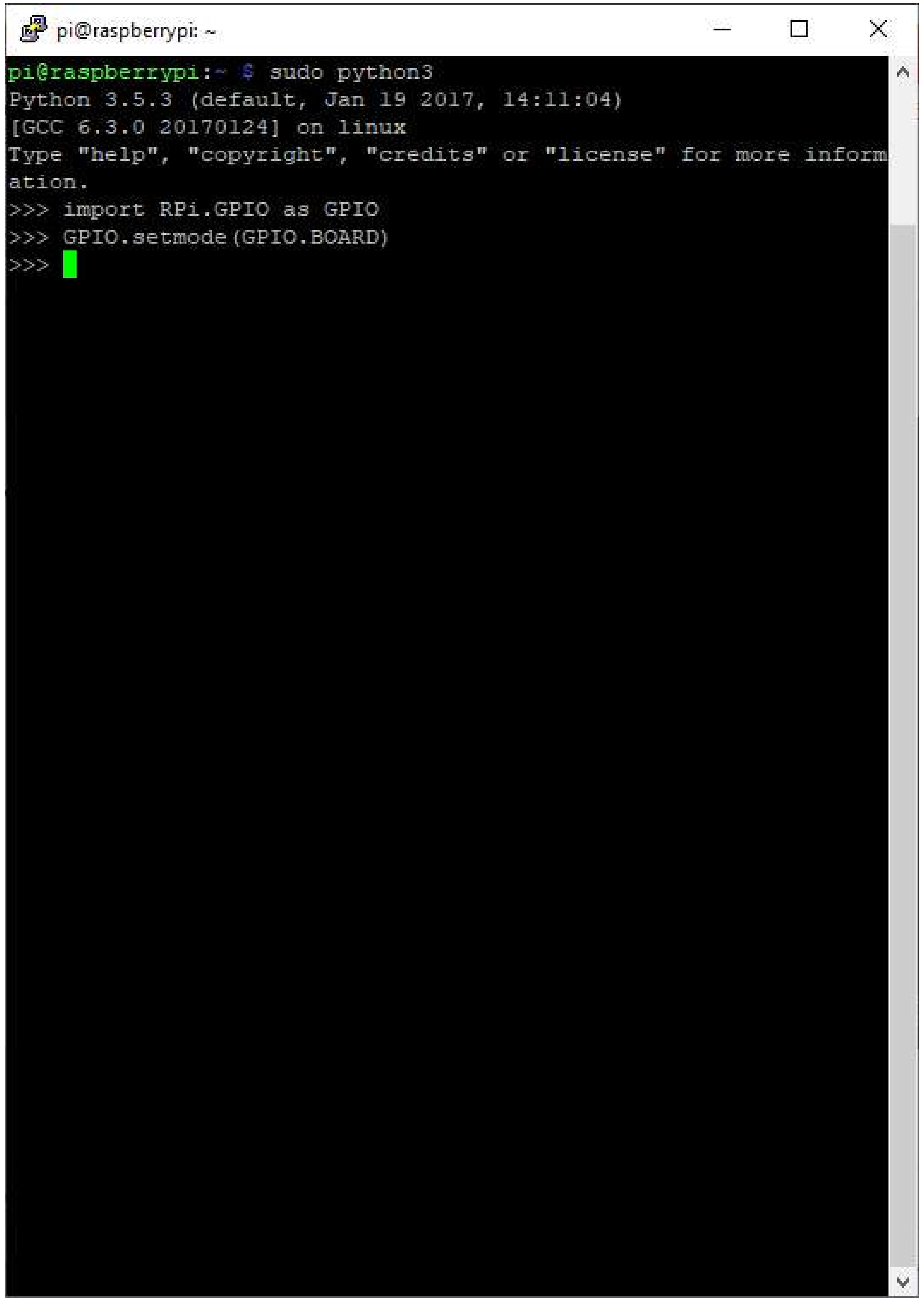
- Import RPi.GPIO library



```
pi@raspberrypi:~ $ sudo python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more inform
ation.
>>> import RPi.GPIO as GPIO
>>>
```

Teleoperation

- Setup GPIO pins

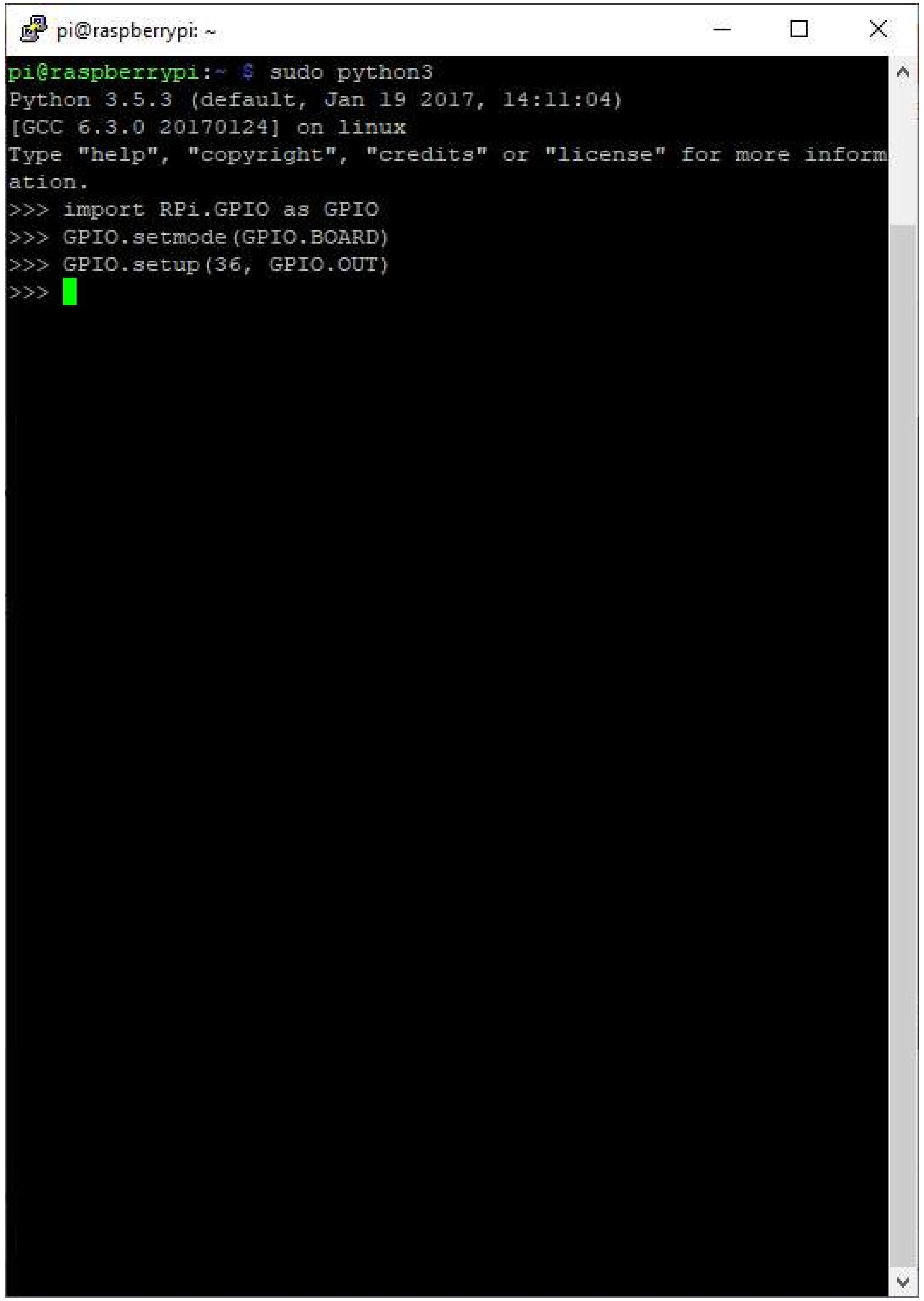


pi@raspberrypi: ~

```
pi@raspberrypi:~ $ sudo python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more inform
ation.
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BOARD)
>>> 
```

Teleoperation

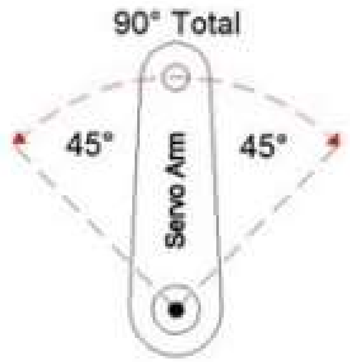
- Set GPIO pin 36 as an output



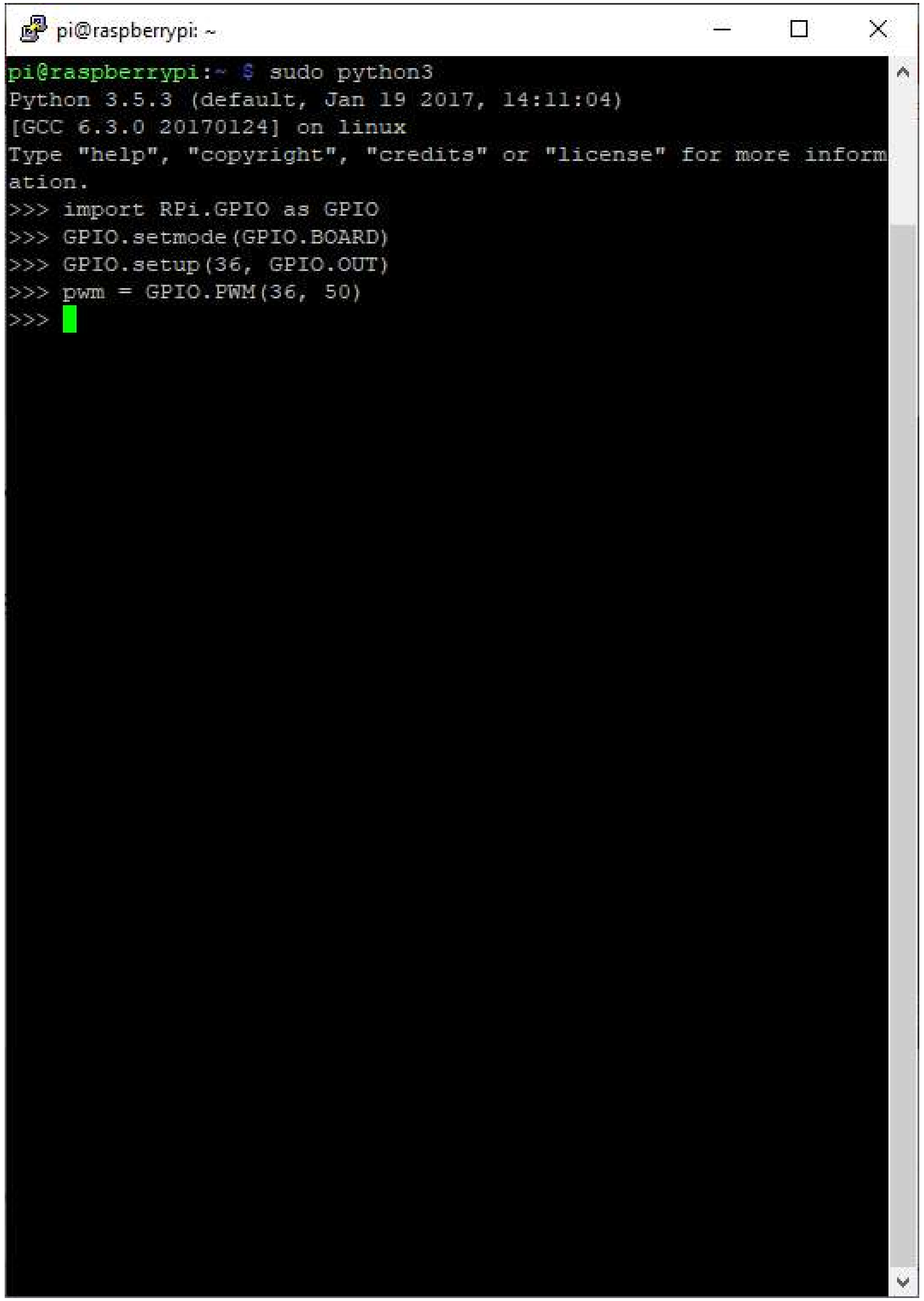
```
pi@raspberrypi:~ $ sudo python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more inform
ation.
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BOARD)
>>> GPIO.setup(36, GPIO.OUT)
>>> 
```

Teleoperation

- Set 50 Hz GPIO output frequency on pin 36



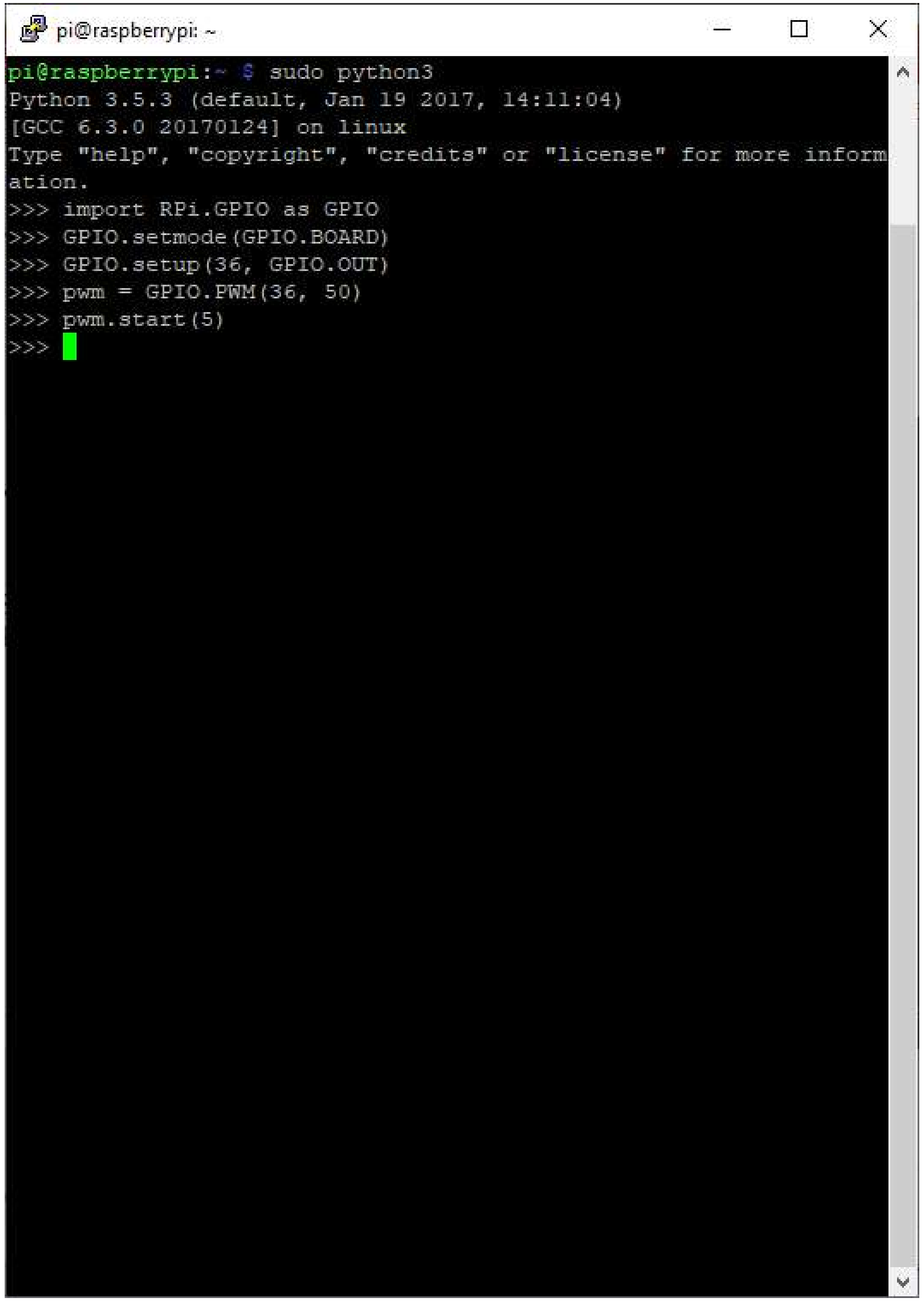
Detailed Specifications
Control System: +Pulse Width Control 1500usec Neutral
Required Pulse: 3-5 Volt Peak to Peak Square Wave



```
pi@raspberrypi:~ $ sudo python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more inform
ation.
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BOARD)
>>> GPIO.setup(36, GPIO.OUT)
>>> pwm = GPIO.PWM(36, 50)
>>> 
```

Teleoperation

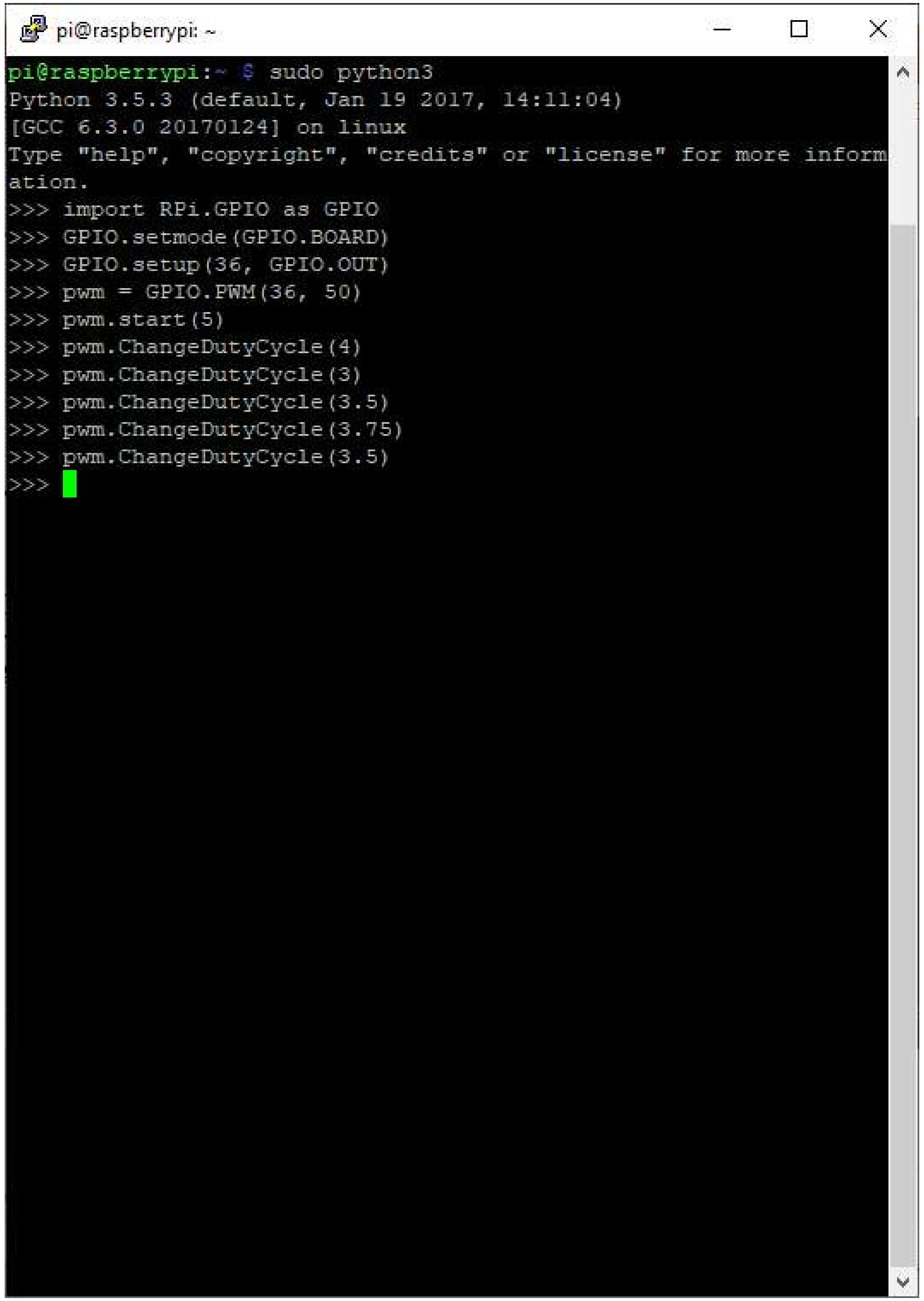
- Start PWM frequency at 5% duty cycle



```
pi@raspberrypi:~ $ sudo python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more inform
ation.
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BOARD)
>>> GPIO.setup(36, GPIO.OUT)
>>> pwm = GPIO.PWM(36, 50)
>>> pwm.start(5)
>>>
```

Teleoperation

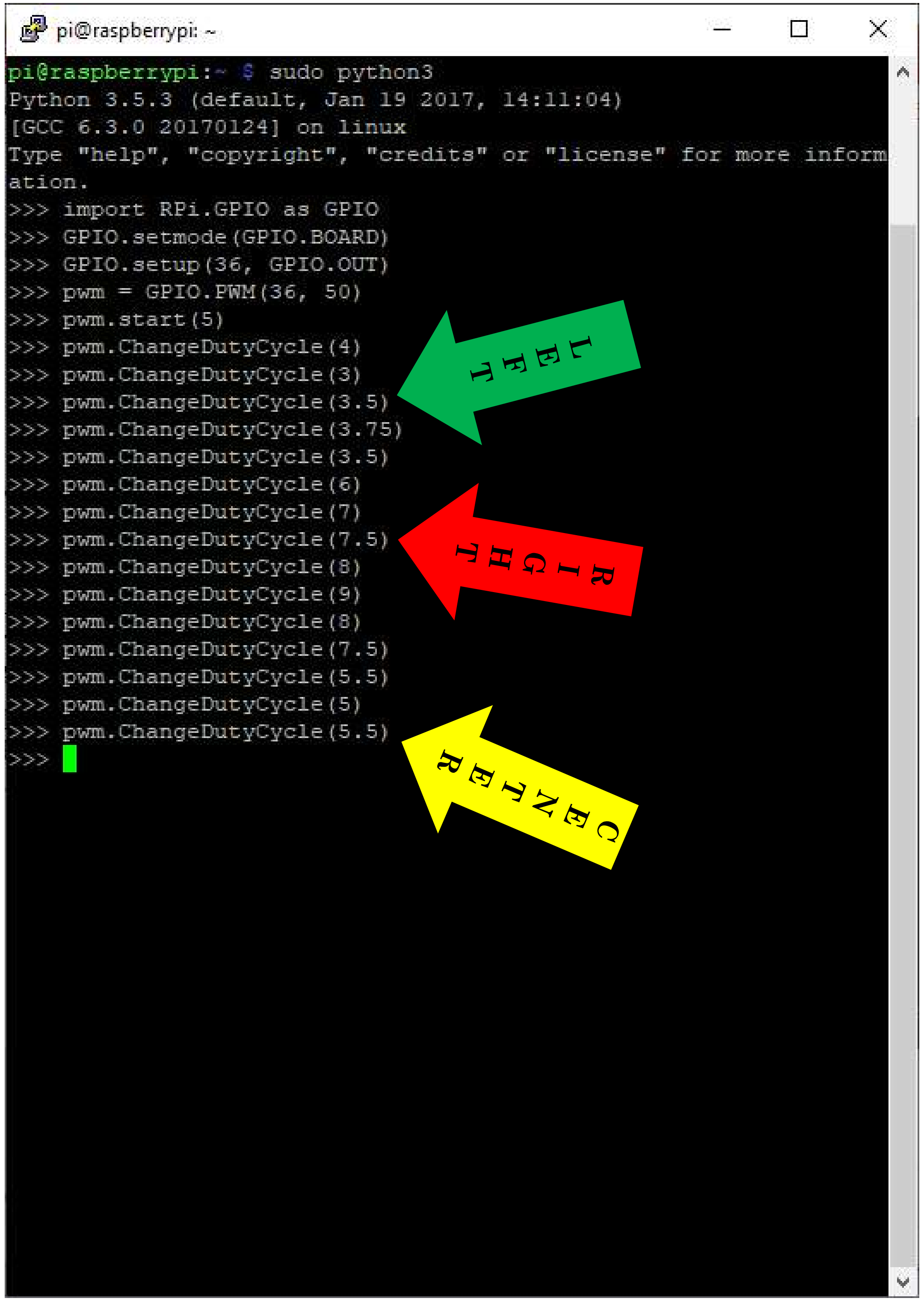
- Change PWM duty cycle as required to achieve range of gripper spacing
- The minimum & maximum duty cycles will be **specific to your servo motor**



```
pi@raspberrypi:~ $ sudo python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more inform
ation.
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BOARD)
>>> GPIO.setup(36, GPIO.OUT)
>>> pwm = GPIO.PWM(36, 50)
>>> pwm.start(5)
>>> pwm.ChangeDutyCycle(4)
>>> pwm.ChangeDutyCycle(3)
>>> pwm.ChangeDutyCycle(3.5)
>>> pwm.ChangeDutyCycle(3.75)
>>> pwm.ChangeDutyCycle(3.5)
>>>
```

Teleoperation

- Change PWM duty cycle as required to achieve range of gripper spacing
- The minimum & maximum duty cycles will be **specific to your servo motor**
- For example:

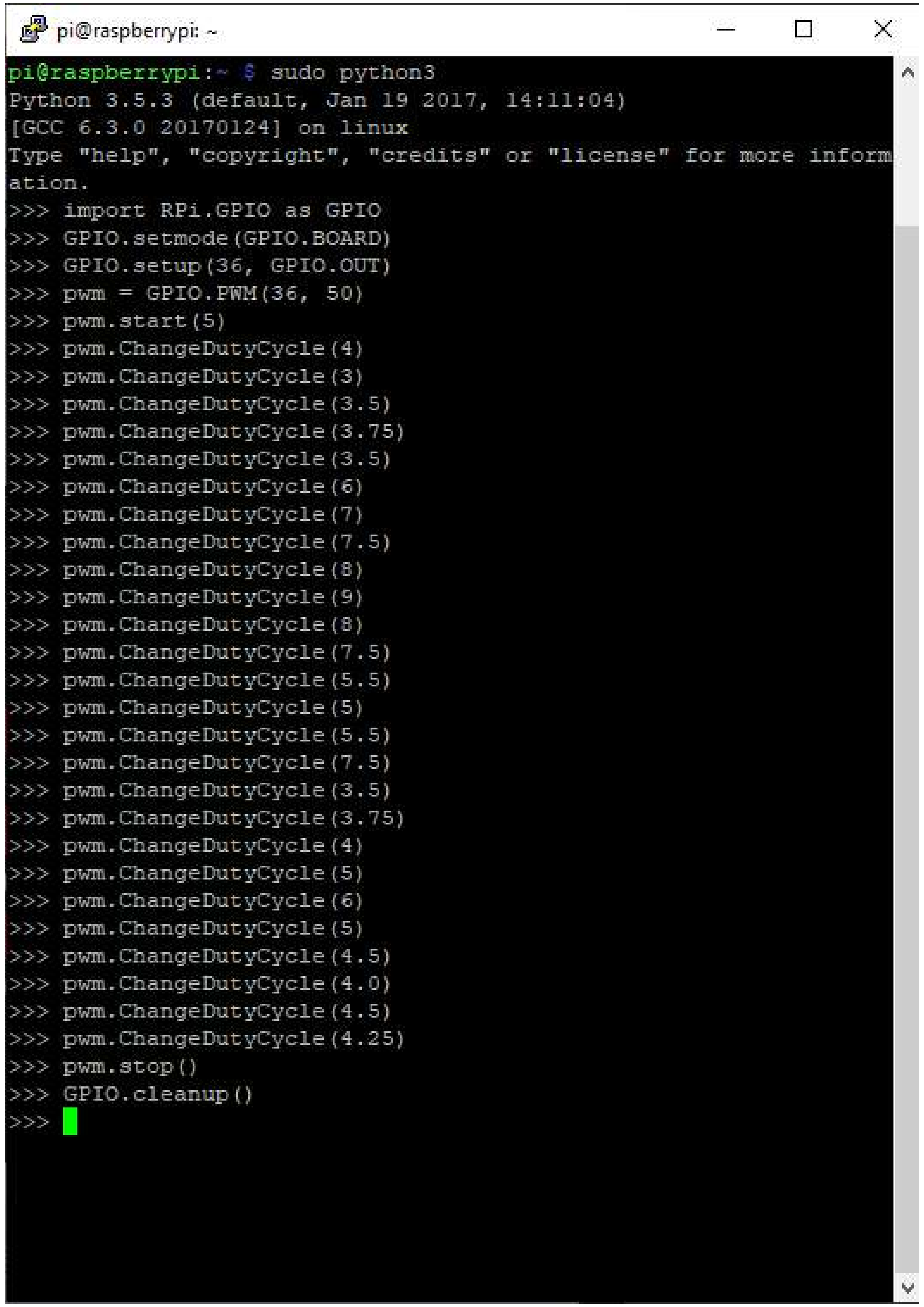


```
pi@raspberrypi:~ $ sudo python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more inform
ation.
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BOARD)
>>> GPIO.setup(36, GPIO.OUT)
>>> pwm = GPIO.PWM(36, 50)
>>> pwm.start(5)
>>> pwm.ChangeDutyCycle(4)
>>> pwm.ChangeDutyCycle(3)
>>> pwm.ChangeDutyCycle(3.5)
>>> pwm.ChangeDutyCycle(3.75)
>>> pwm.ChangeDutyCycle(3.5)
>>> pwm.ChangeDutyCycle(6)
>>> pwm.ChangeDutyCycle(7)
>>> pwm.ChangeDutyCycle(7.5)
>>> pwm.ChangeDutyCycle(8)
>>> pwm.ChangeDutyCycle(9)
>>> pwm.ChangeDutyCycle(8)
>>> pwm.ChangeDutyCycle(7.5)
>>> pwm.ChangeDutyCycle(5.5)
>>> pwm.ChangeDutyCycle(5)
>>> pwm.ChangeDutyCycle(5.5)
>>>
```

Teleoperation

- Again, it is vitally important **not to exceed the upper limit** on duty cycle
- When finished:

1. Stop PWM output
2. Cleanup the GPIO pins



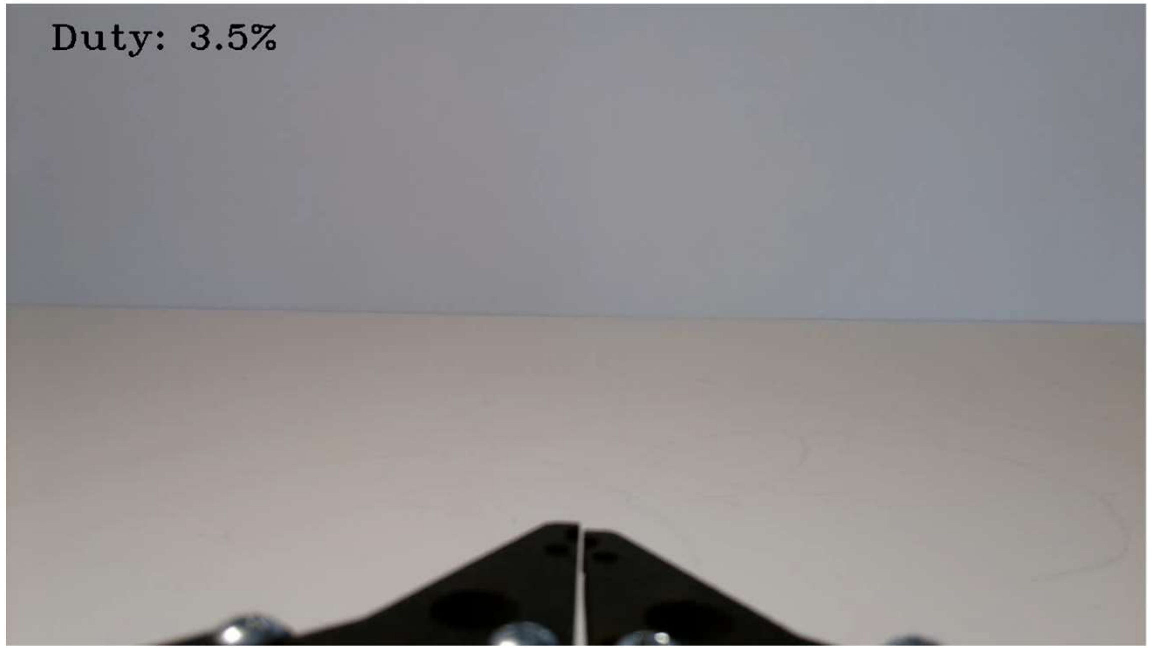
```
pi@raspberrypi:~ $ sudo python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more inform
ation.
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BOARD)
>>> GPIO.setup(36, GPIO.OUT)
>>> pwm = GPIO.PWM(36, 50)
>>> pwm.start(5)
>>> pwm.ChangeDutyCycle(4)
>>> pwm.ChangeDutyCycle(3)
>>> pwm.ChangeDutyCycle(3.5)
>>> pwm.ChangeDutyCycle(3.75)
>>> pwm.ChangeDutyCycle(3.5)
>>> pwm.ChangeDutyCycle(6)
>>> pwm.ChangeDutyCycle(7)
>>> pwm.ChangeDutyCycle(7.5)
>>> pwm.ChangeDutyCycle(8)
>>> pwm.ChangeDutyCycle(9)
>>> pwm.ChangeDutyCycle(8)
>>> pwm.ChangeDutyCycle(7.5)
>>> pwm.ChangeDutyCycle(5.5)
>>> pwm.ChangeDutyCycle(5)
>>> pwm.ChangeDutyCycle(5.5)
>>> pwm.ChangeDutyCycle(7.5)
>>> pwm.ChangeDutyCycle(3.5)
>>> pwm.ChangeDutyCycle(3.75)
>>> pwm.ChangeDutyCycle(4)
>>> pwm.ChangeDutyCycle(5)
>>> pwm.ChangeDutyCycle(6)
>>> pwm.ChangeDutyCycle(5)
>>> pwm.ChangeDutyCycle(4.5)
>>> pwm.ChangeDutyCycle(4.0)
>>> pwm.ChangeDutyCycle(4.5)
>>> pwm.ChangeDutyCycle(4.25)
>>> pwm.stop()
>>> GPIO.cleanup()
>>>
```

In-Class Exercise

- Create new Python script: ***servocontrol01.py***
- When executed, script must:

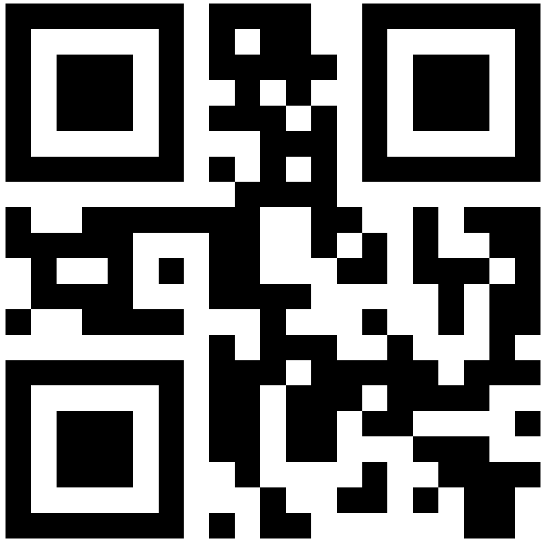
1. Slowly (user-define "slowly") cycle gripper from open to closed and back again
2. Record an image with the RPi camera at each gripper position
3. Print duty cycle onto each image
4. Stich images together to generate time-lapse video

**** Ensure your script cannot exceed the lower & upper bounds on duty cycle***



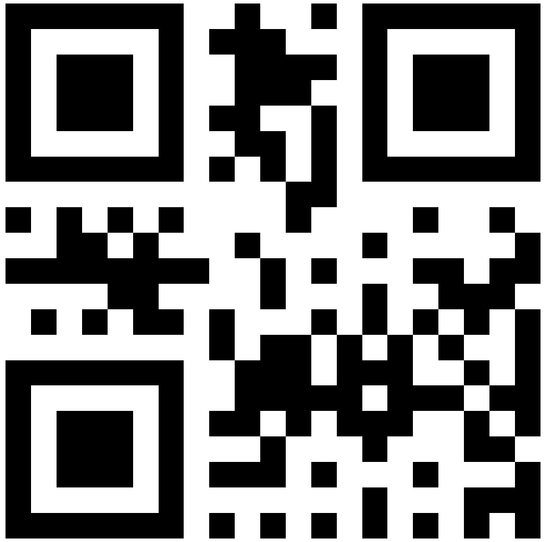
Case Study: Robotic Control

3.5%



"CLOSE"

5.5%



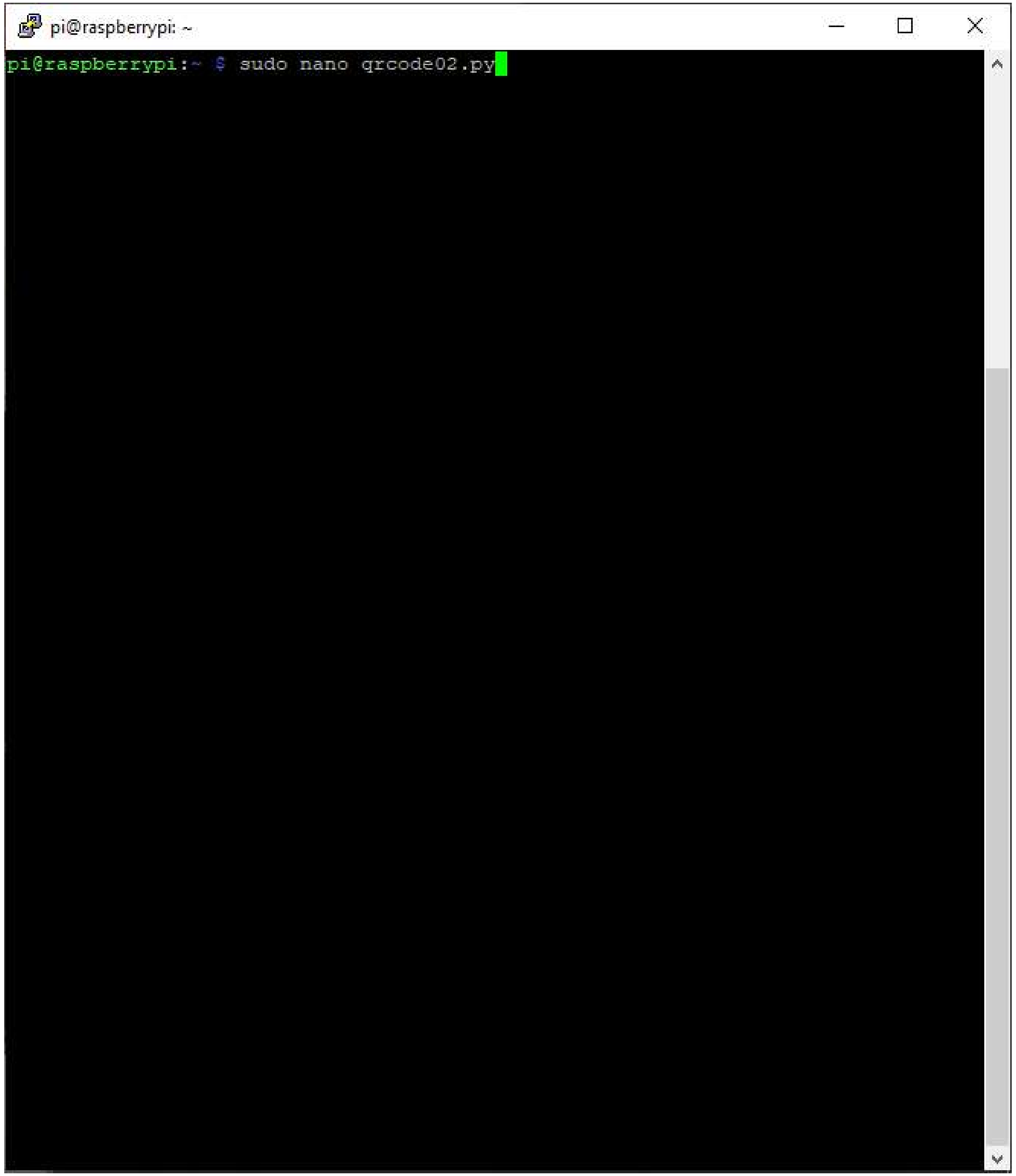
"HALF"

7.5%

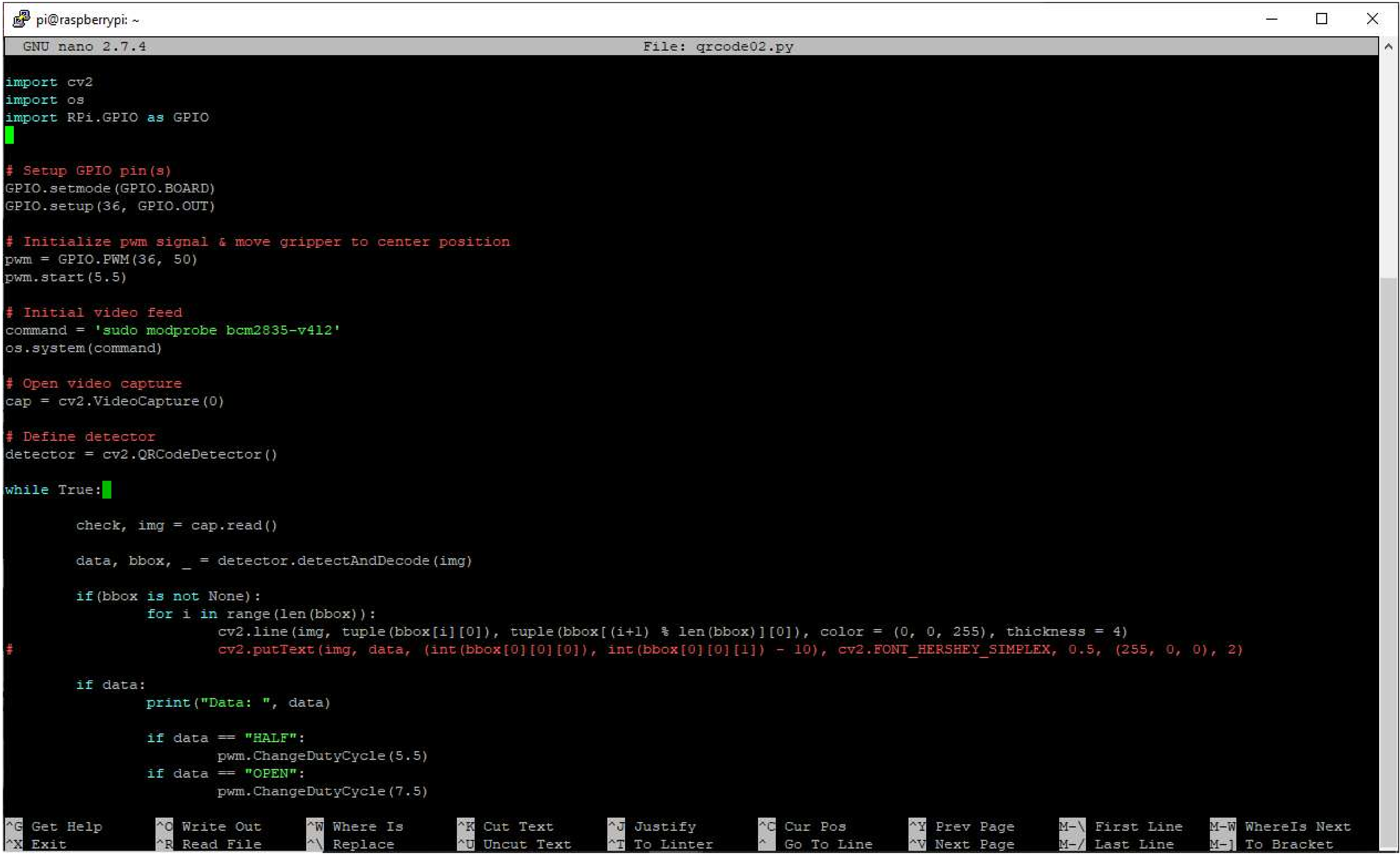
"OPEN"

Raspberry Pi

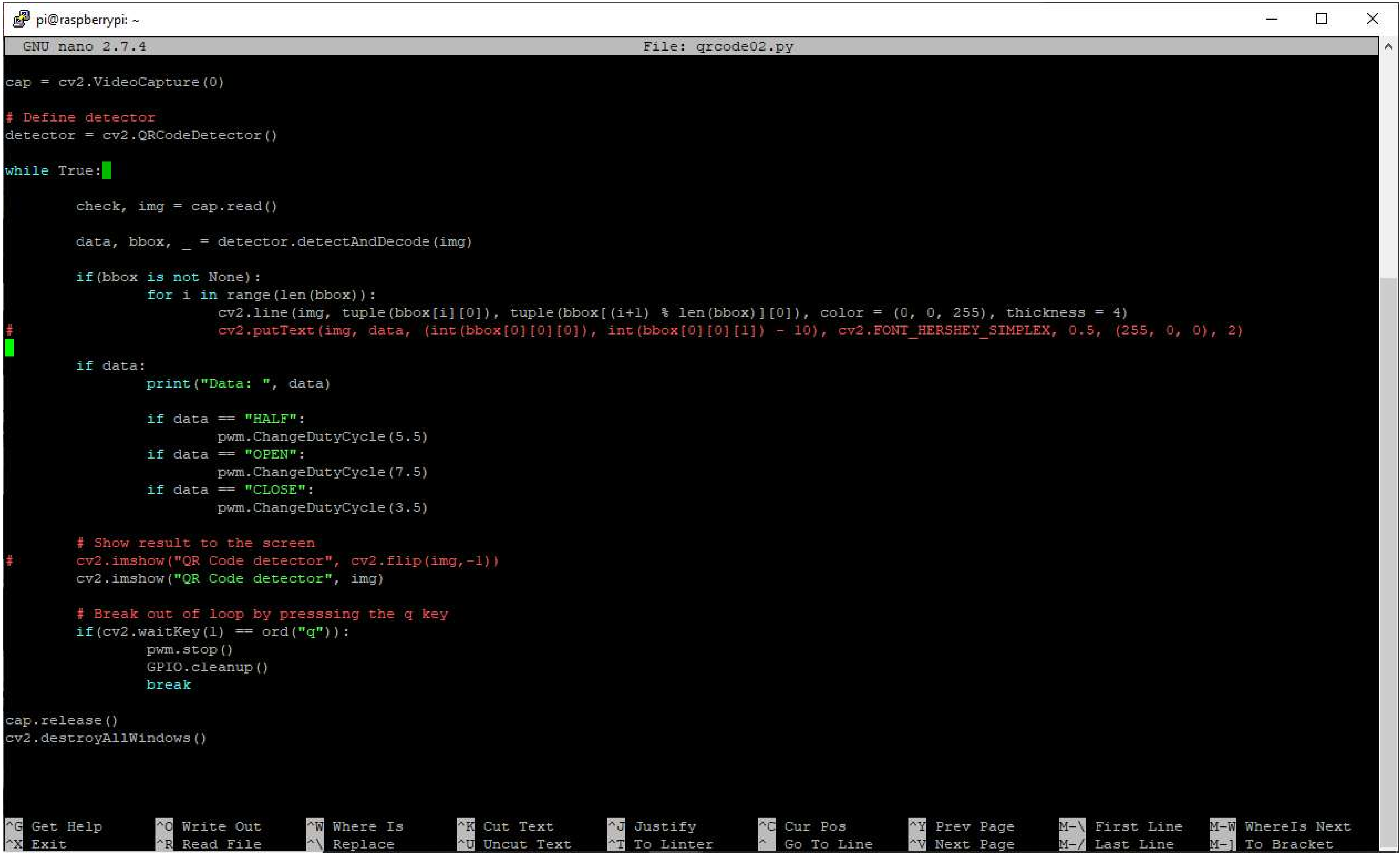
- Copy & create script ***qrcode02.py***



```
pi@raspberrypi:~ $ sudo nano qrcode02.py
```

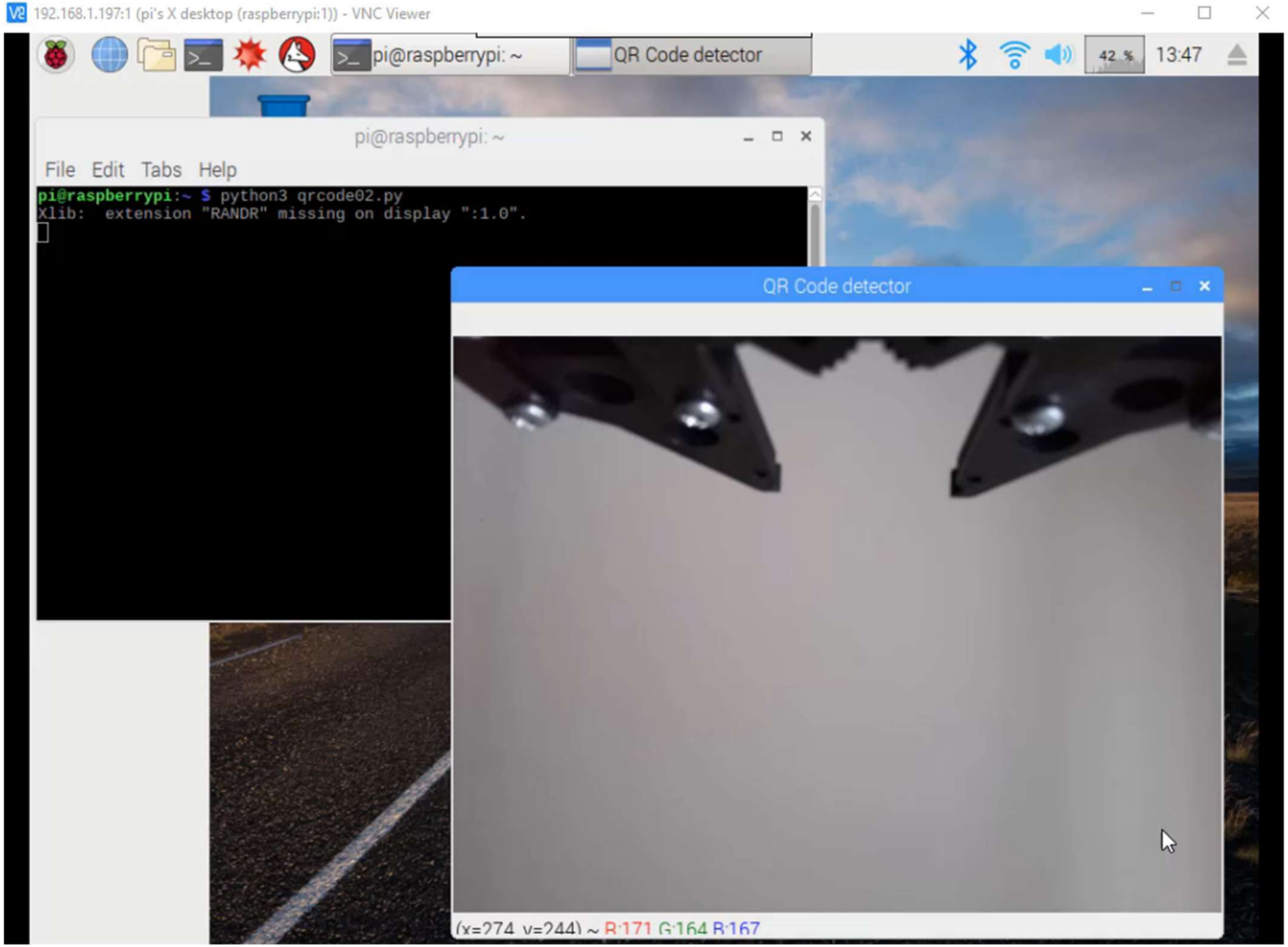


```
import cv2
import os
import RPi.GPIO as GPIO


# Setup GPIO pin(s)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(36, GPIO.OUT)

# Initialize pwm signal & move gripper to center position
pwm = GPIO.PWM(36, 50)
pwm.start(5.5)

# Initial video feed
command = 'sudo modprobe bcm2835-v4l2'
os.system(command)

# Open video capture
cap = cv2.VideoCapture(0)

# Define detector
detector = cv2.QRCodeDetector()

while True:

        check, img = cap.read()

        data, bbox, _ = detector.detectAndDecode(img)

        if(bbox is not None):
                for i in range(len(bbox)):
                        cv2.line(img, tuple(bbox[i][0]), tuple(bbox[(i+1) % len(bbox)][0]), color = (0, 0, 255), thickness = 4)
#                       cv2.putText(img, data, (int(bbox[0][0][0]), int(bbox[0][0][1]) - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

        if data:
                print("Data: ", data)

                if data == "HALF":
                        pwm.ChangeDutyCycle(5.5)
                if data == "OPEN":
                        pwm.ChangeDutyCycle(7.5)
```



pi@raspberrypi: ~

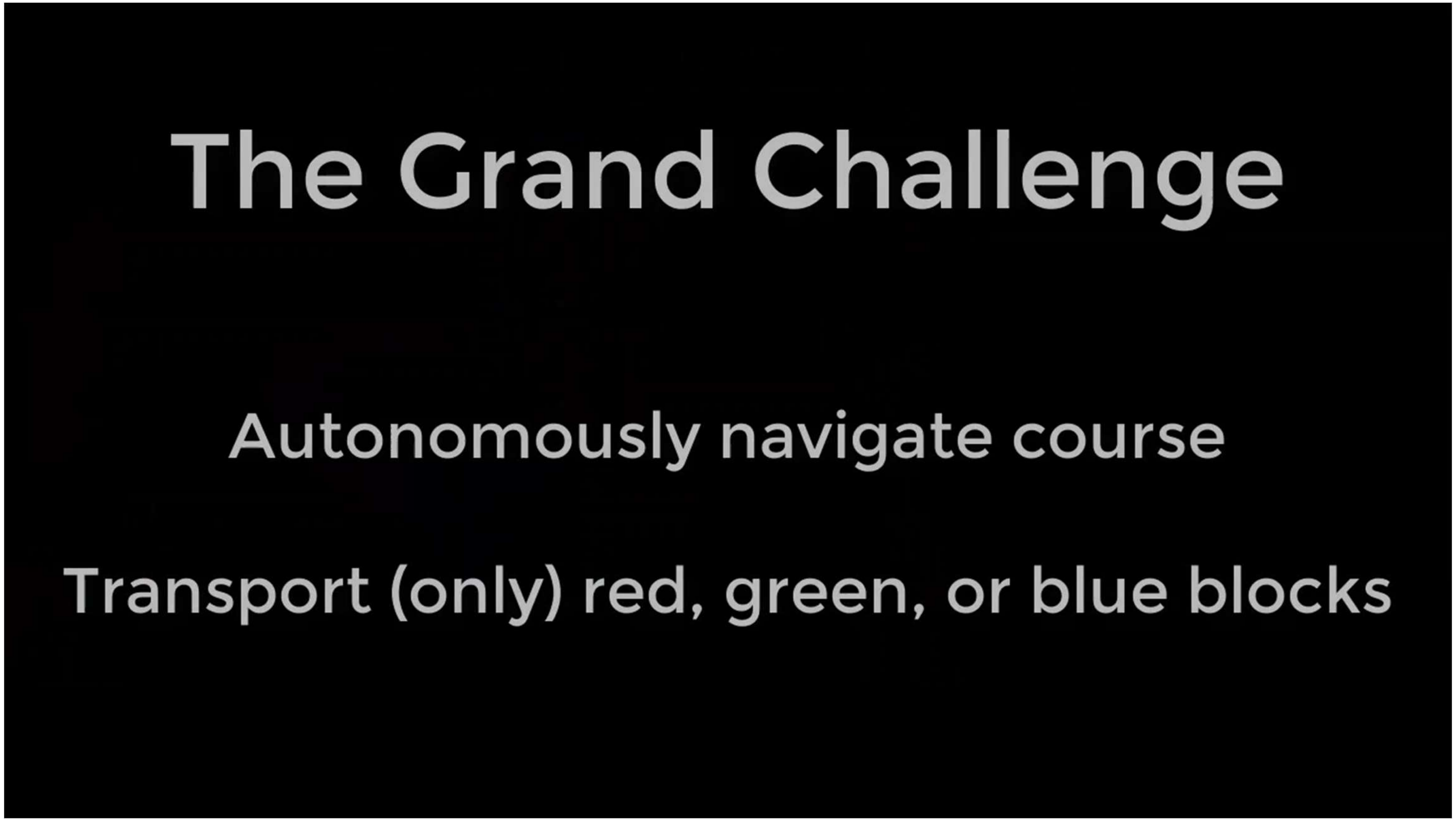
```
  GNU nano 2.7.4                               File: qrcode02.py

cap = cv2.VideoCapture(0)

# Define detector
detector = cv2.QRCodeDetector()

while True:

        check, img = cap.read()

        data, bbox, _ = detector.detectAndDecode(img)

        if(bbox is not None):
                for i in range(len(bbox)):
                        cv2.line(img, tuple(bbox[i][0]), tuple(bbox[(i+1) % len(bbox)][0]), color = (0, 0, 255), thickness = 4)
#                       cv2.putText(img, data, (int(bbox[0][0][0]), int(bbox[0][0][1]) - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

        if data:
                print("Data: ", data)

                if data == "HALF":
                        pwm.ChangeDutyCycle(5.5)
                if data == "OPEN":
                        pwm.ChangeDutyCycle(7.5)
                if data == "CLOSE":
                        pwm.ChangeDutyCycle(3.5)

        # Show result to the screen
#       cv2.imshow("QR Code detector", cv2.flip(img,-1))
        cv2.imshow("QR Code detector", img)

        # Break out of loop by presssing the q key
        if(cv2.waitKey(1) == ord("q")):
                pwm.stop()
                GPIO.cleanup()
                break

cap.release()
cv2.destroyAllWindows()

^G Get Help     ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos      ^Y Prev Page    M-\ First Line  M-W WhereIs Next
^X Exit         ^R Read File    ^\ Replace      ^U Uncut Text   ^T To Linter    ^_ Go To Line   ^V Next Page    M-/ Last Line   M-] To Bracket
```



```
pi@raspberrypi:~ $ python3 qrcode02.py
Xlib:  extension "RANDR" missing on display ":1.0".
```

(x=274, y=244) ~ R:171 G:164 B:167

Case Study: Robotic Control



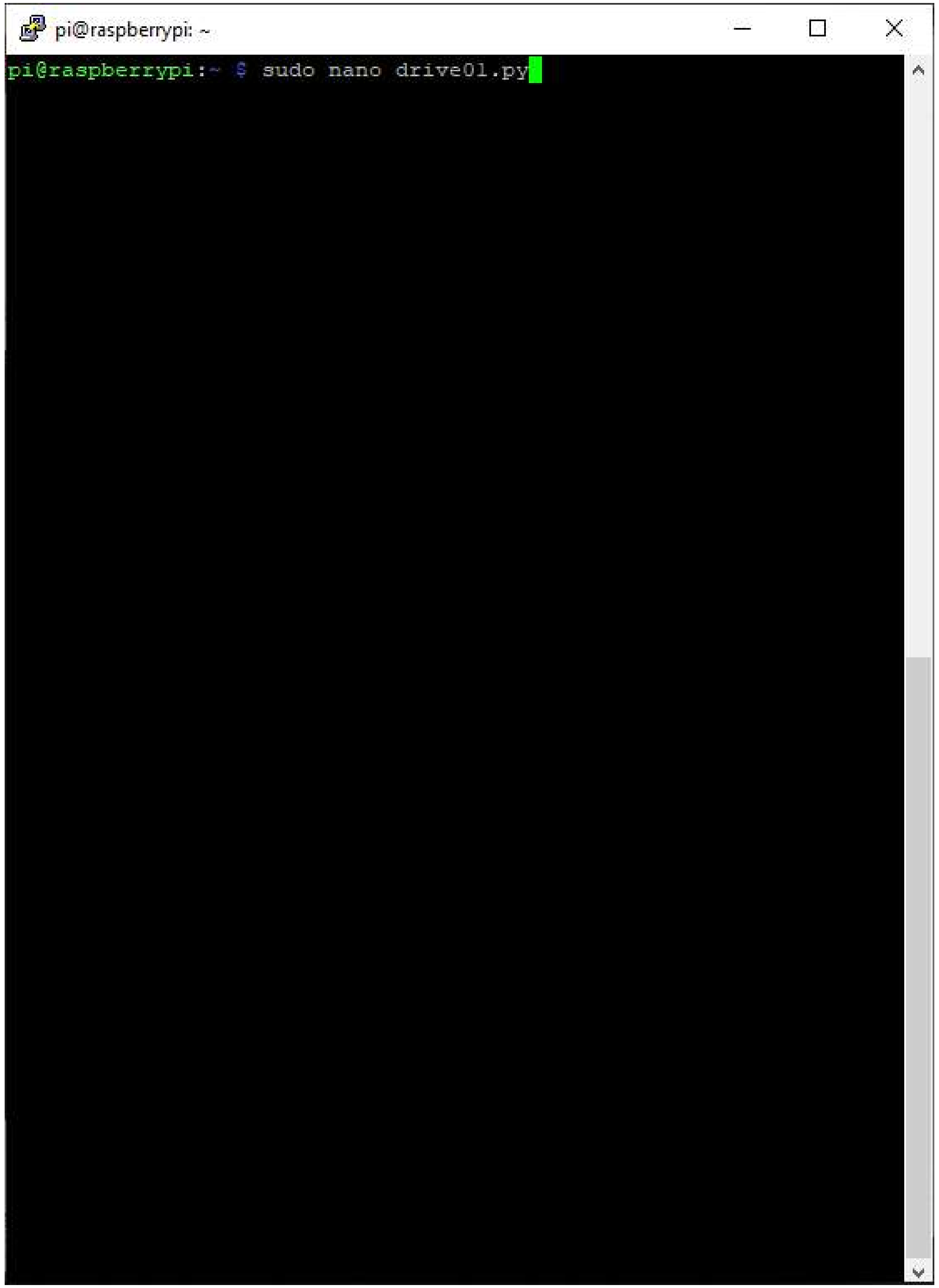
The Grand Challenge

Autonomously navigate course

Transport (only) red, green, or blue blocks

Teleoperation

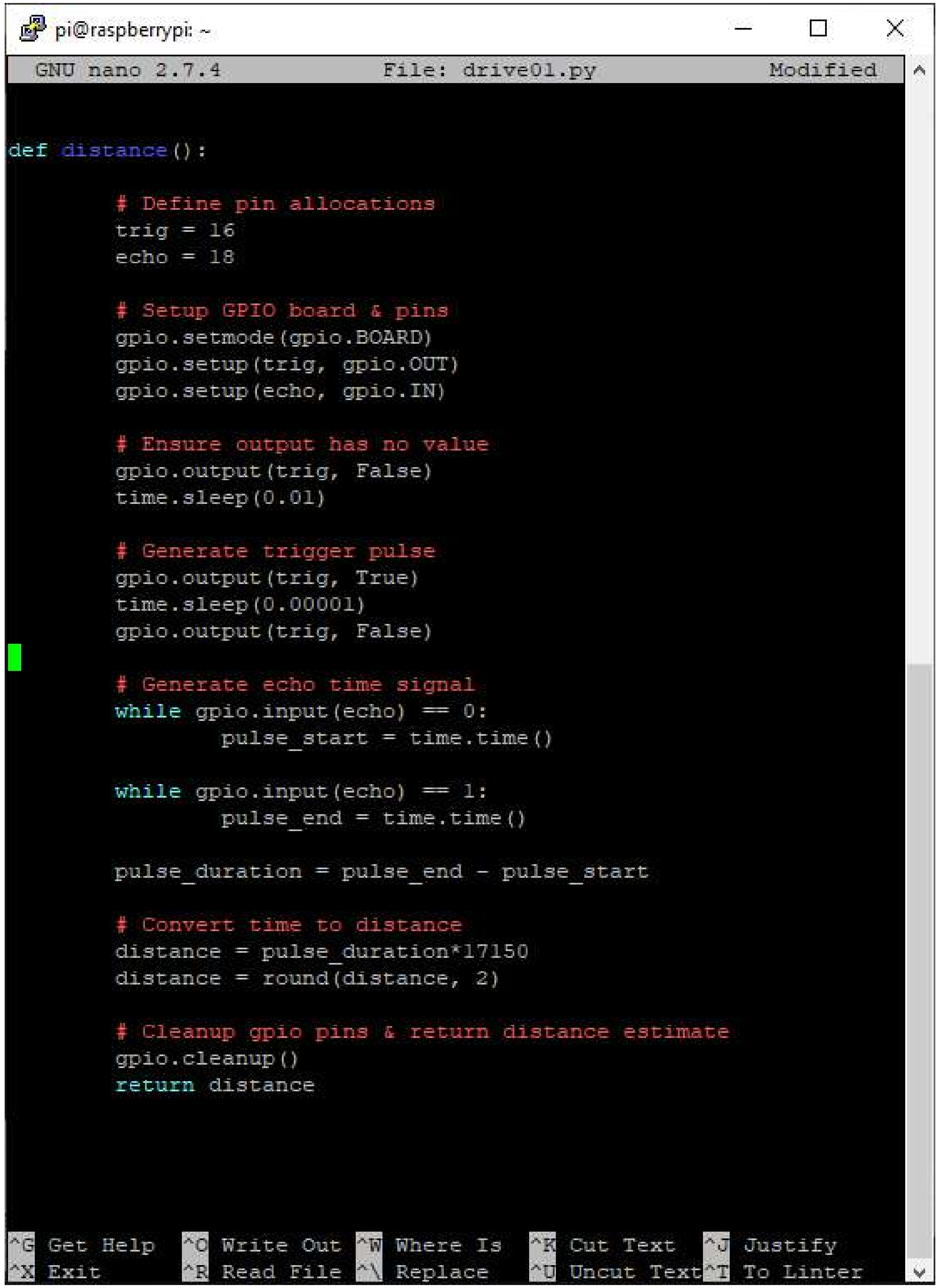
- Create a new Python script: ***drive01.py***



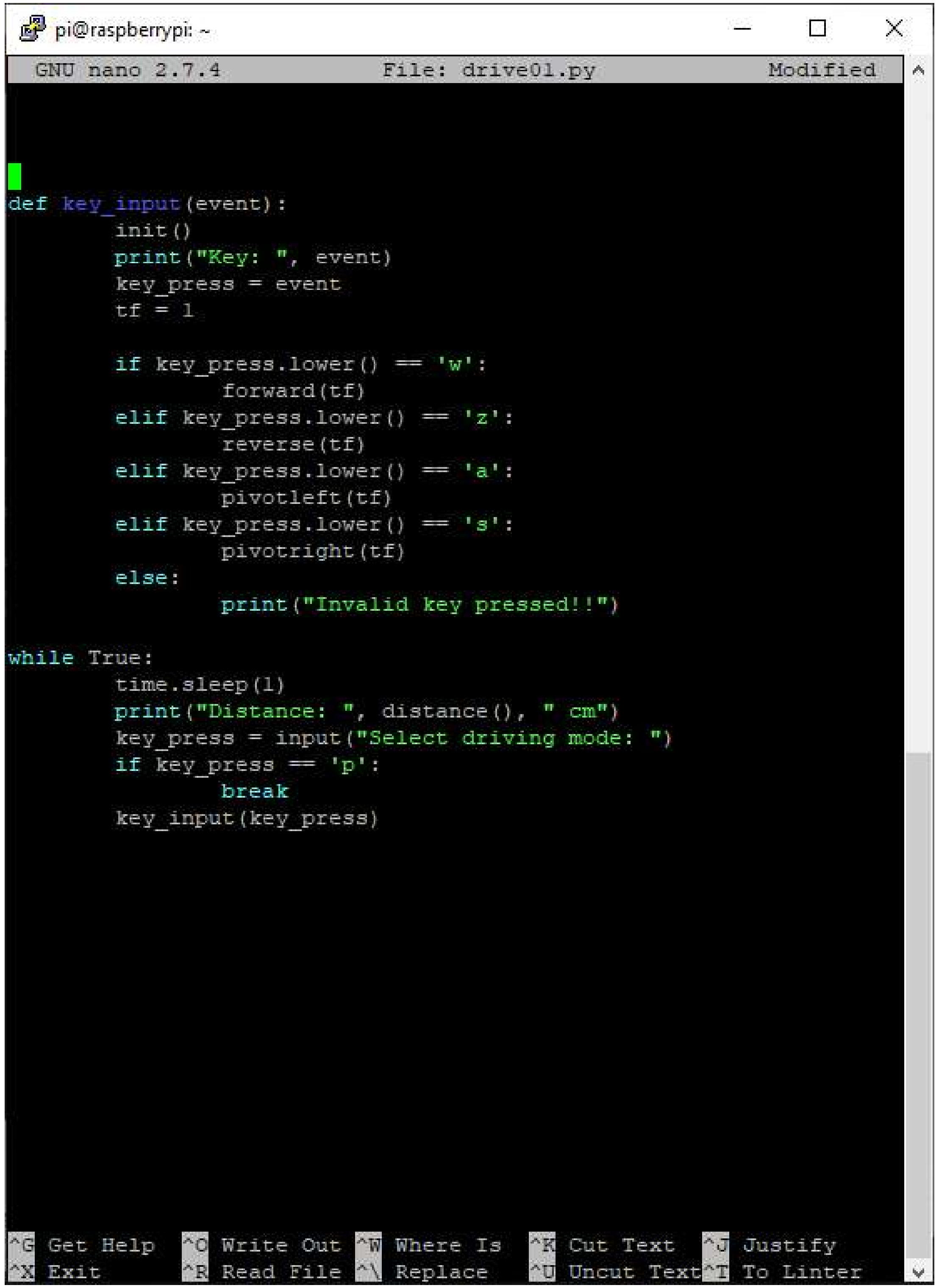
```
pi@raspberrypi:~ $ sudo nano drive01.py
```

Teleoperation

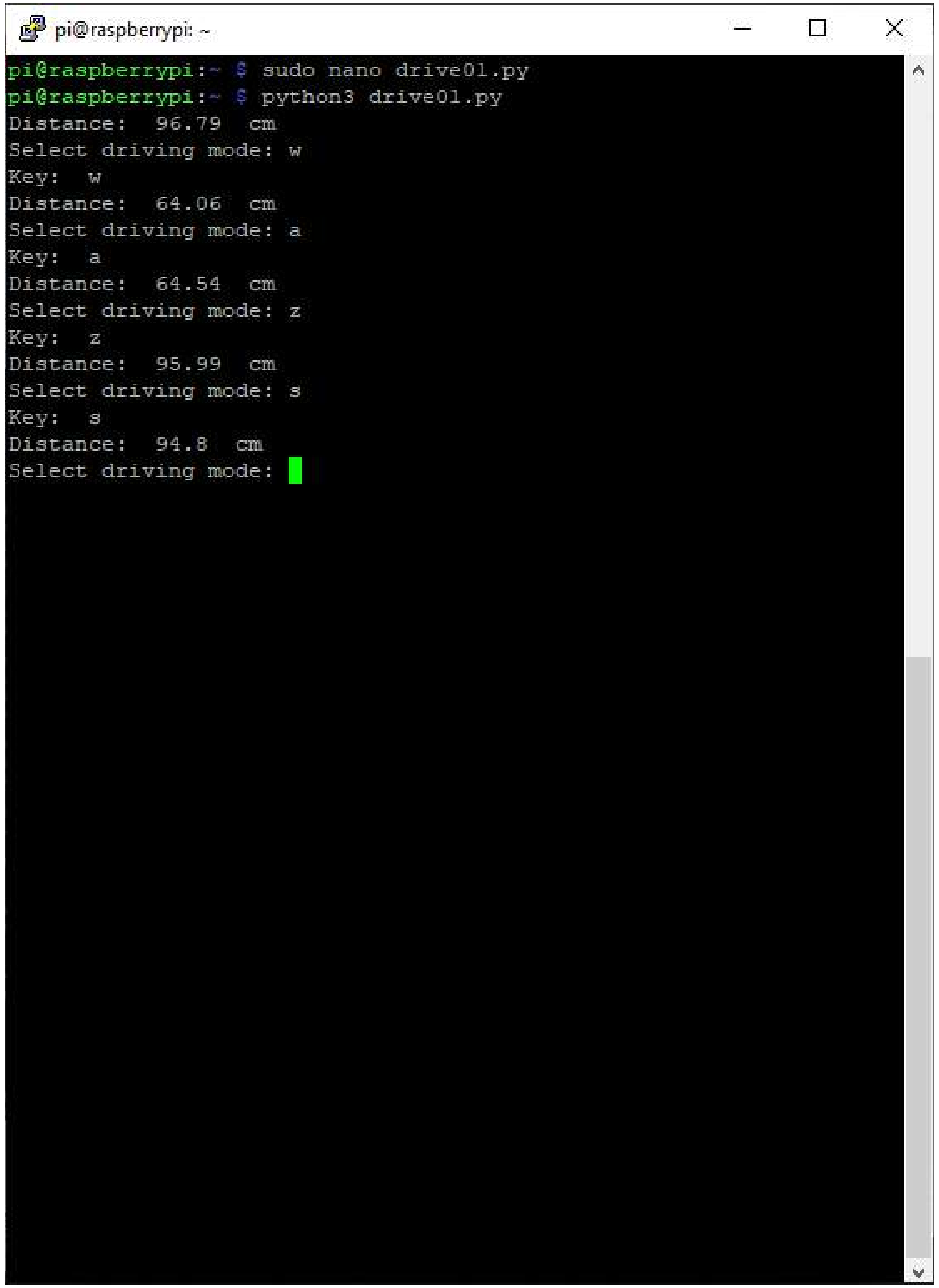
- Create a new Python script: ***drive01.py***
- Add collision avoidance functionality to Python using sodar sensor
  - Utilize script from Assignment #4
- May require tilting sonar sensor or mounting sonar breadboard on spacer



```
def distance():

        # Define pin allocations
        trig = 16
        echo = 18

        # Setup GPIO board & pins
        gpio.setmode(gpio.BOARD)
        gpio.setup(trig, gpio.OUT)
        gpio.setup(echo, gpio.IN)

        # Ensure output has no value
        gpio.output(trig, False)
        time.sleep(0.01)

        # Generate trigger pulse
        gpio.output(trig, True)
        time.sleep(0.00001)
        gpio.output(trig, False)

        # Generate echo time signal
        while gpio.input(echo) == 0:
                pulse_start = time.time()

        while gpio.input(echo) == 1:
                pulse_end = time.time()

        pulse_duration = pulse_end - pulse_start

        # Convert time to distance
        distance = pulse_duration*17150
        distance = round(distance, 2)

        # Cleanup gpio pins & return distance estimate
        gpio.cleanup()
        return distance
```



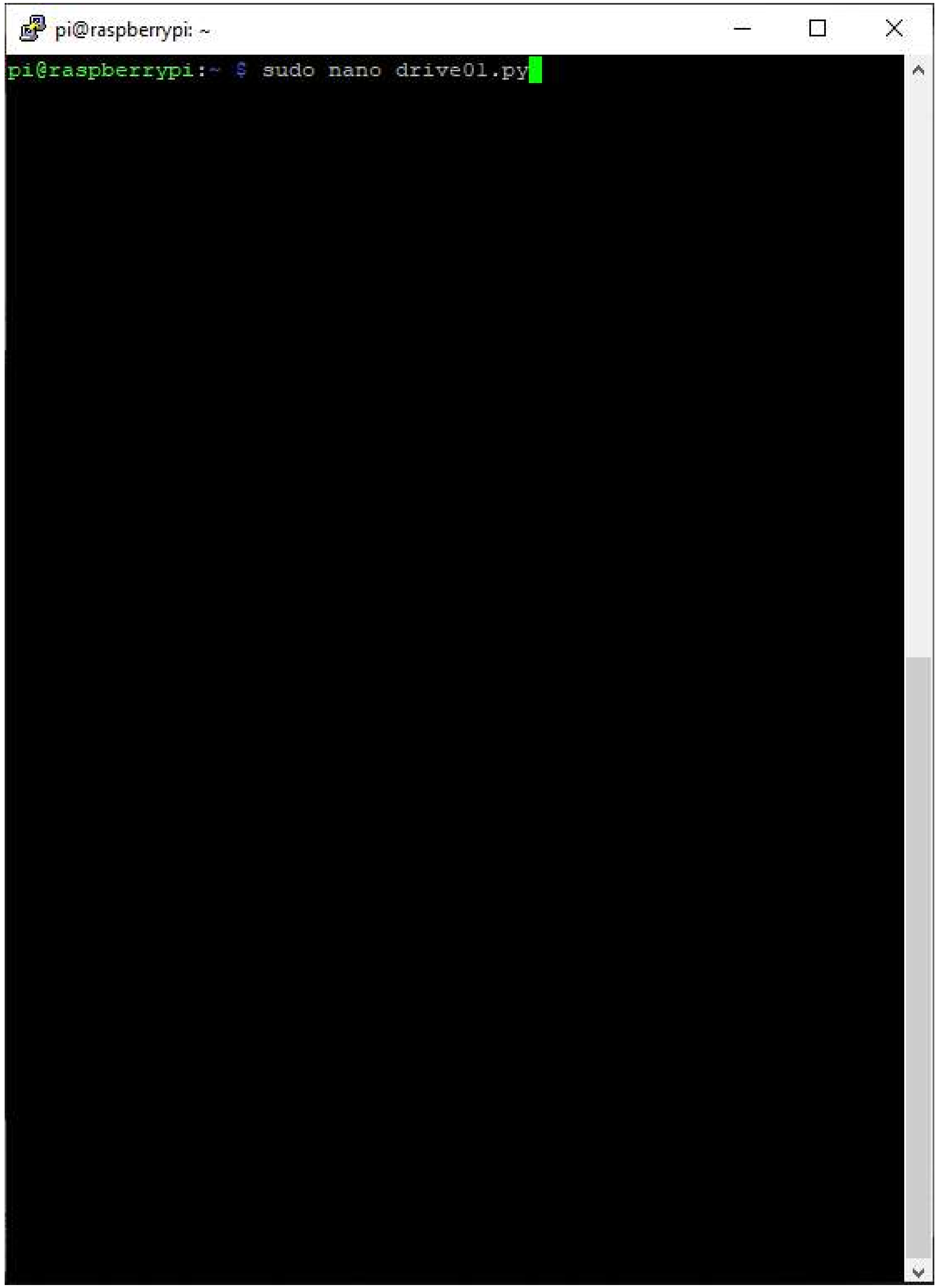
```


def key_input(event):
        init()
        print("Key: ", event)
        key_press = event
        tf = 1

        if key_press.lower() == 'w':
                forward(tf)
        elif key_press.lower() == 'z':
                reverse(tf)
        elif key_press.lower() == 'a':
                pivotleft(tf)
        elif key_press.lower() == 's':
                pivotright(tf)
        else:
                print("Invalid key pressed!!")

while True:
        time.sleep(1)
        print("Distance: ", distance(), " cm")
        key_press = input("Select driving mode: ")
        if key_press == 'p':
                break
        key_input(key_press)
```



```
pi@raspberrypi:~ $ sudo nano drive01.py
pi@raspberrypi:~ $ python3 drive01.py
Distance:  96.79  cm
Select driving mode: w
Key:  w
Distance:  64.06  cm
Select driving mode: a
Key:  a
Distance:  64.54  cm
Select driving mode: z
Key:  z
Distance:  95.99  cm
Select driving mode: s
Key:  s
Distance:  94.8  cm
Select driving mode: 
```

Teleoperation

- Add servo functionality to ***drive01.py***



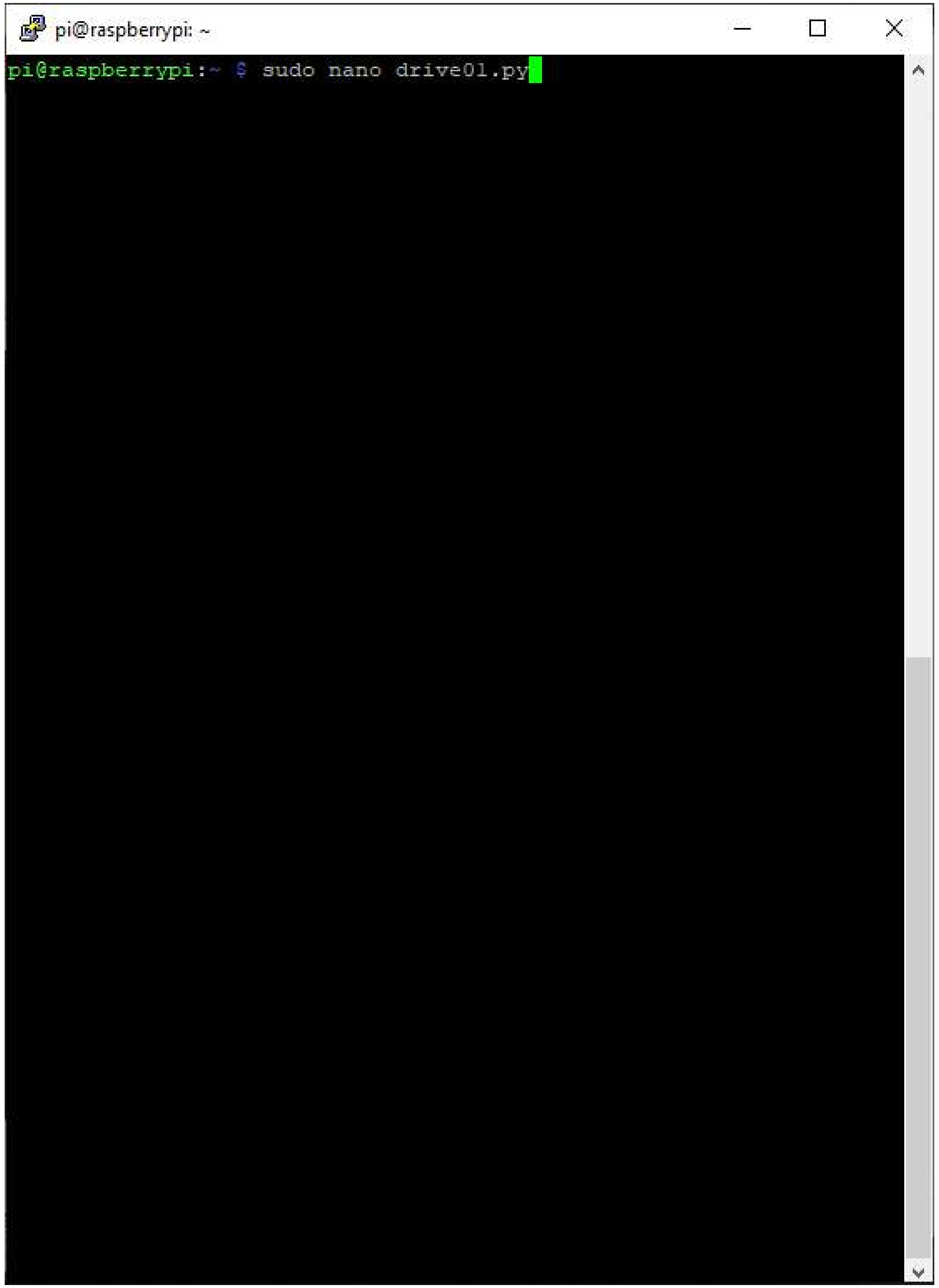
```
pi@raspberrypi:~ $ sudo nano drive01.py
```

Teleoperation

- Add servo functionality to ***drive01.py***
- Take as inputs from the user:

1. Servo duty cycle
2. Drive direction

- **Reject** user input that exceeds the upper duty cycle bound



```
pi@raspberrypi:~ $ sudo nano drive01.py
```

In-Class Exercise

- Add RPi camera functionality to ***drive01.py***
- Create time-lapse video of traversing course
- Video should demonstrate servo functionality as well

References

- *Introduction to Autonomous Mobile Robots*, Siegwart
  - Chapter 2, 3
- *Parallel Gripper Kit A*, Servo City
  - https://www.servocity.com/parallel-gripper-kit-a
- Rpi.GPIO
  - https://pypi.org/project/RPi.GPIO/
- Tutorial – L298N Dual Motor Controller
  - https://hackerstore.nl/PDFs/Tutorial298.pdf
- Servo PWM: Why 20 ms period?
  - https://electronics.stackexchange.com/questions/397715/servo-pwm-why-20-ms-period