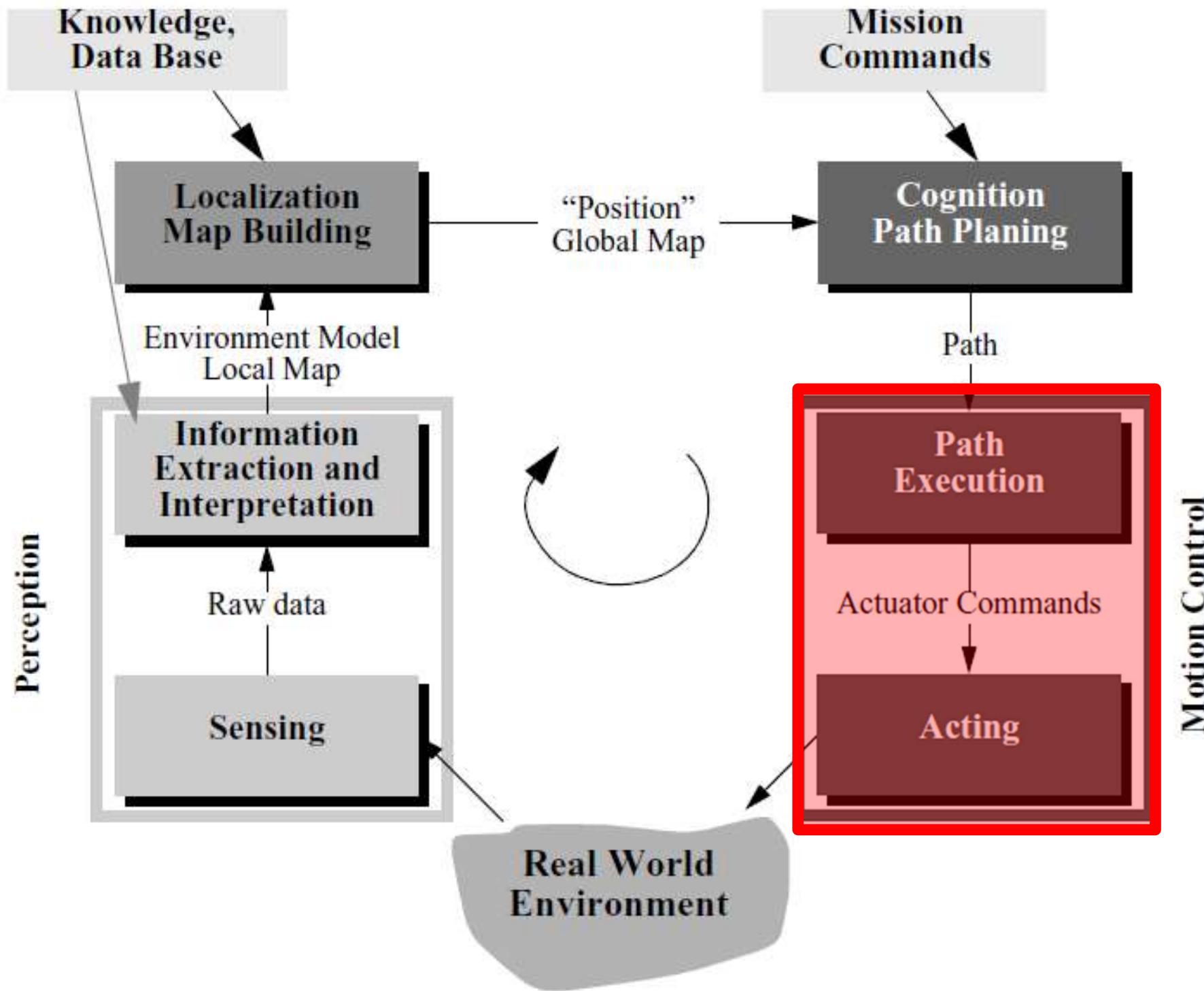


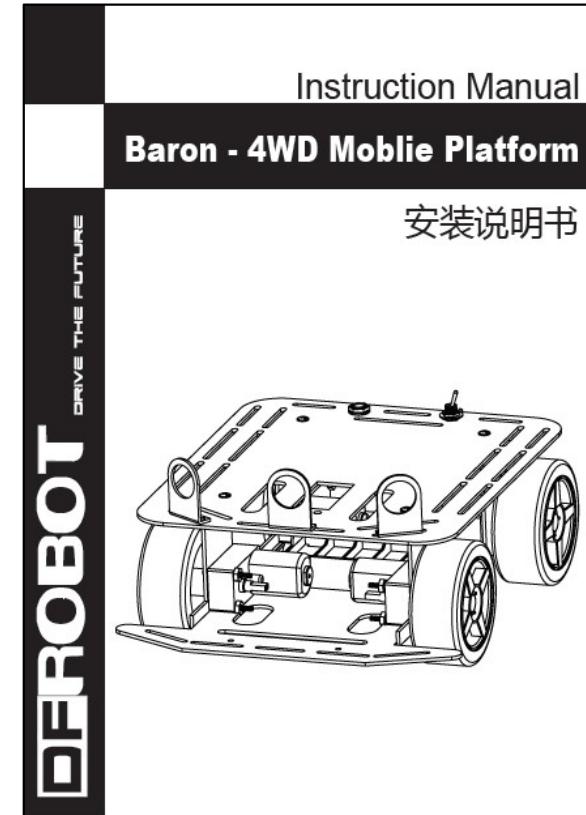
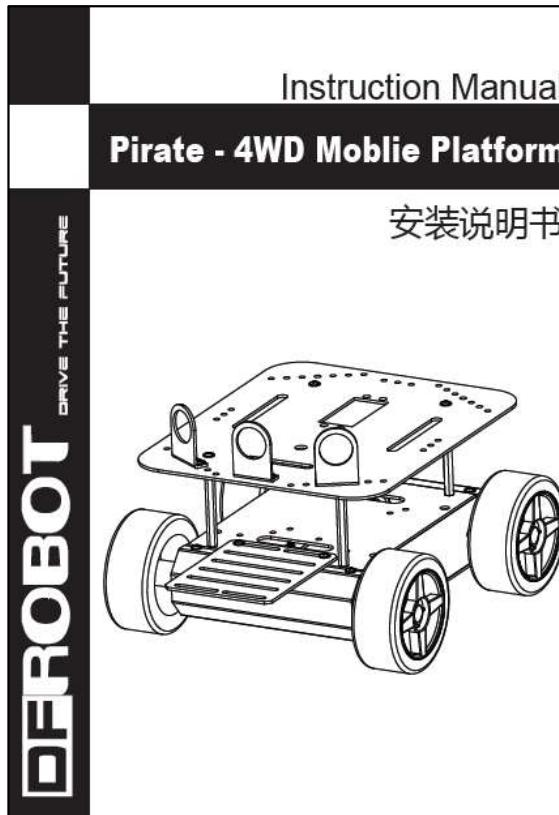
ENPM 809T

UMCP, Mitchell



Assembly

- Materials required to assemble ground vehicle:
 1. Robot kit
 2. Zip ties
 3. 80-deg 3D printed camera mount
 4. Teflon standoffs
 5. Miscellaneous wires
 6. Tools



Assembly

- Unpack materials



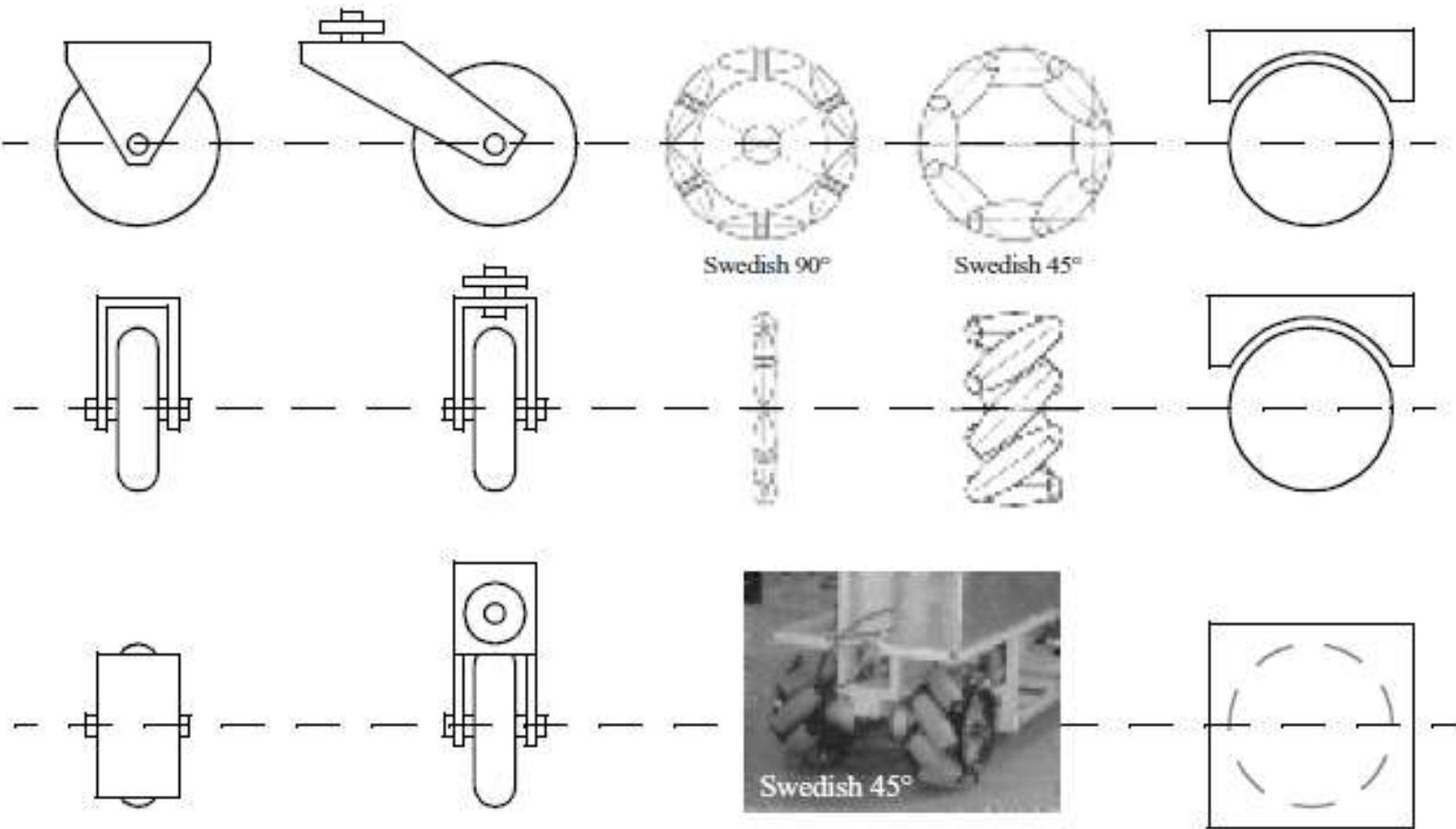
Locomotion

- Mechanisms that enable mobile robots to move unbounded throughout their environment
- Often inspired by biological systems

| Type of motion | Resistance to motion | Basic kinematics of motion |
|-------------------|------------------------|---|
| Flow in a Channel | Hydrodynamic forces | Eddies |
| Crawl | Friction forces | Longitudinal vibration |
| Sliding | Friction forces | Transverse vibration |
| Running | Loss of kinetic energy | Oscillatory movement of a multi-link pendulum |
| Jumping | Loss of kinetic energy | Oscillatory movement of a multi-link pendulum |
| Walking | Gravitational forces | Rolling of a polygon (see figure 2.2) |

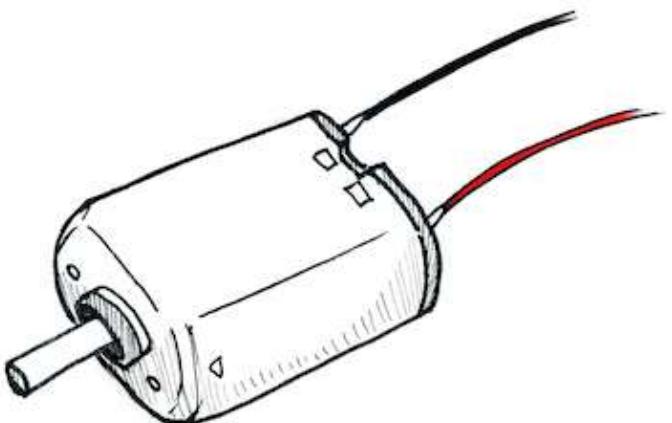
Locomotion

- Actively powered wheel achieves high efficiency on flat ground



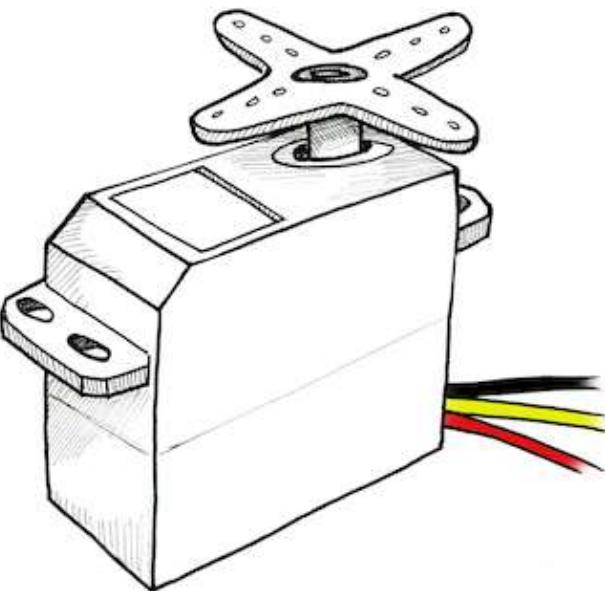
Locomotion

DC MOTOR



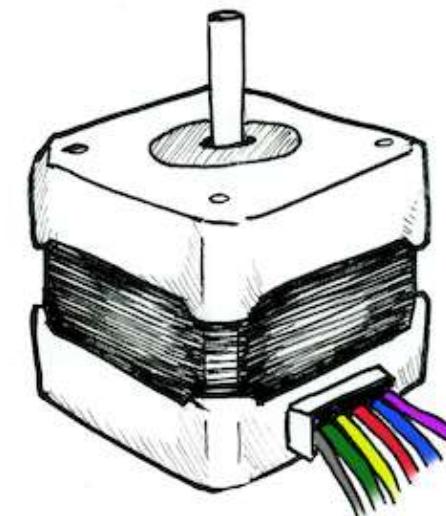
2 WIRES

SERVO MOTOR

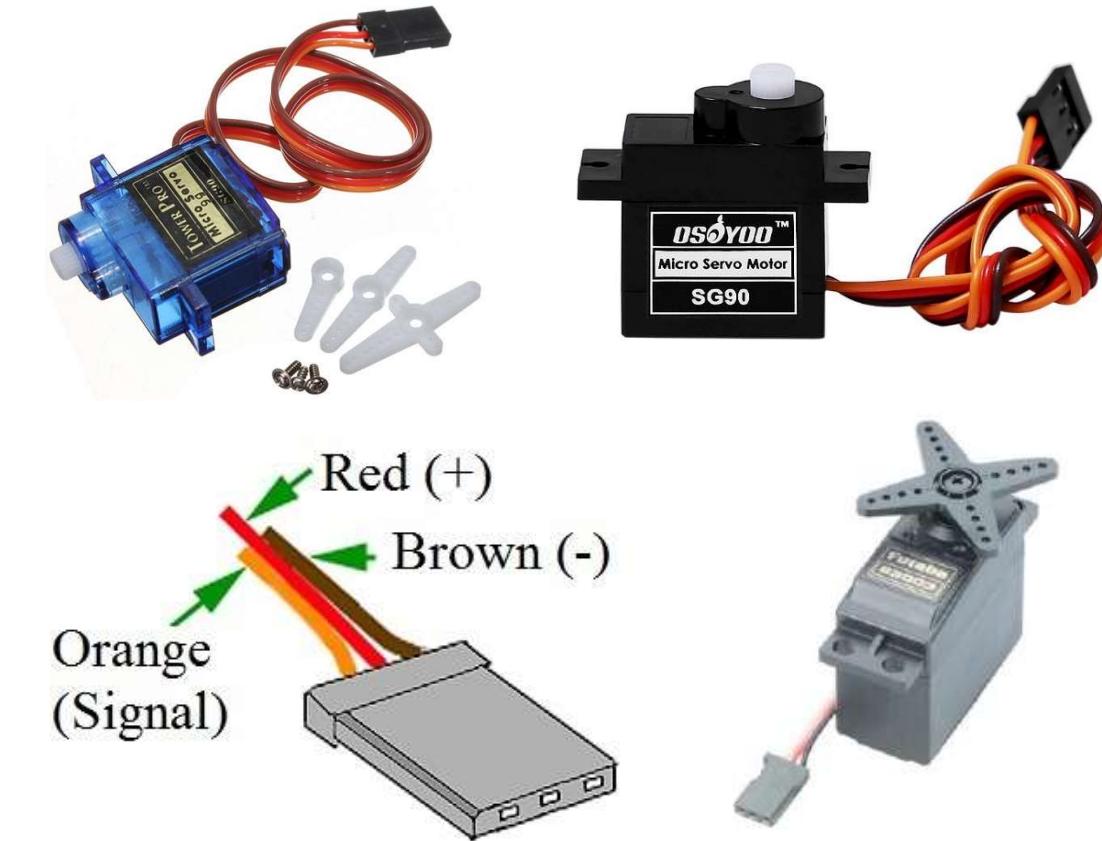
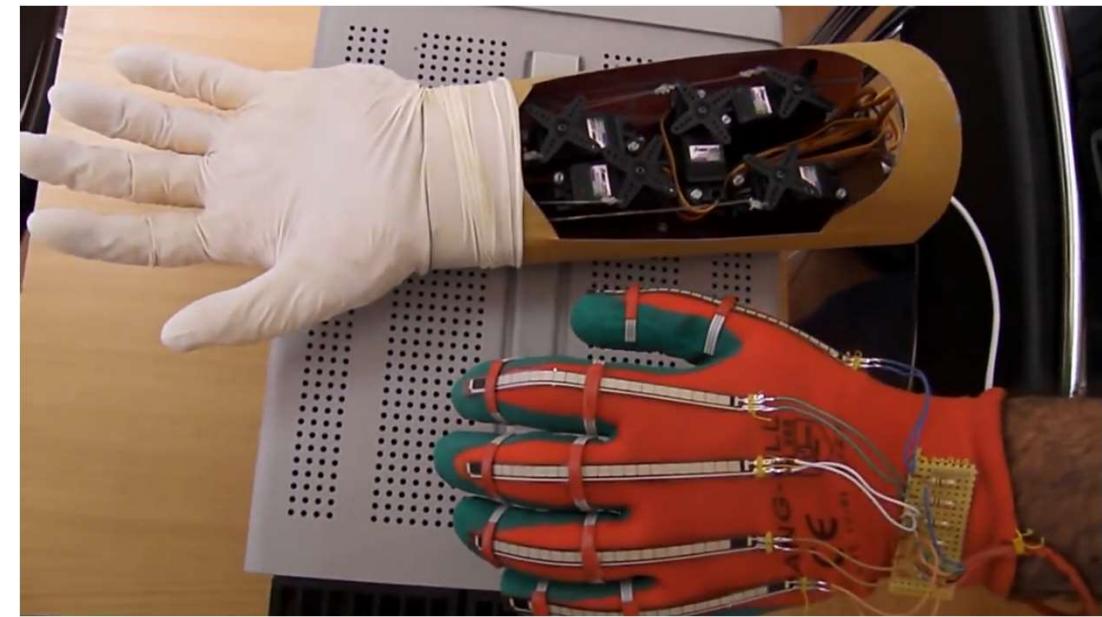
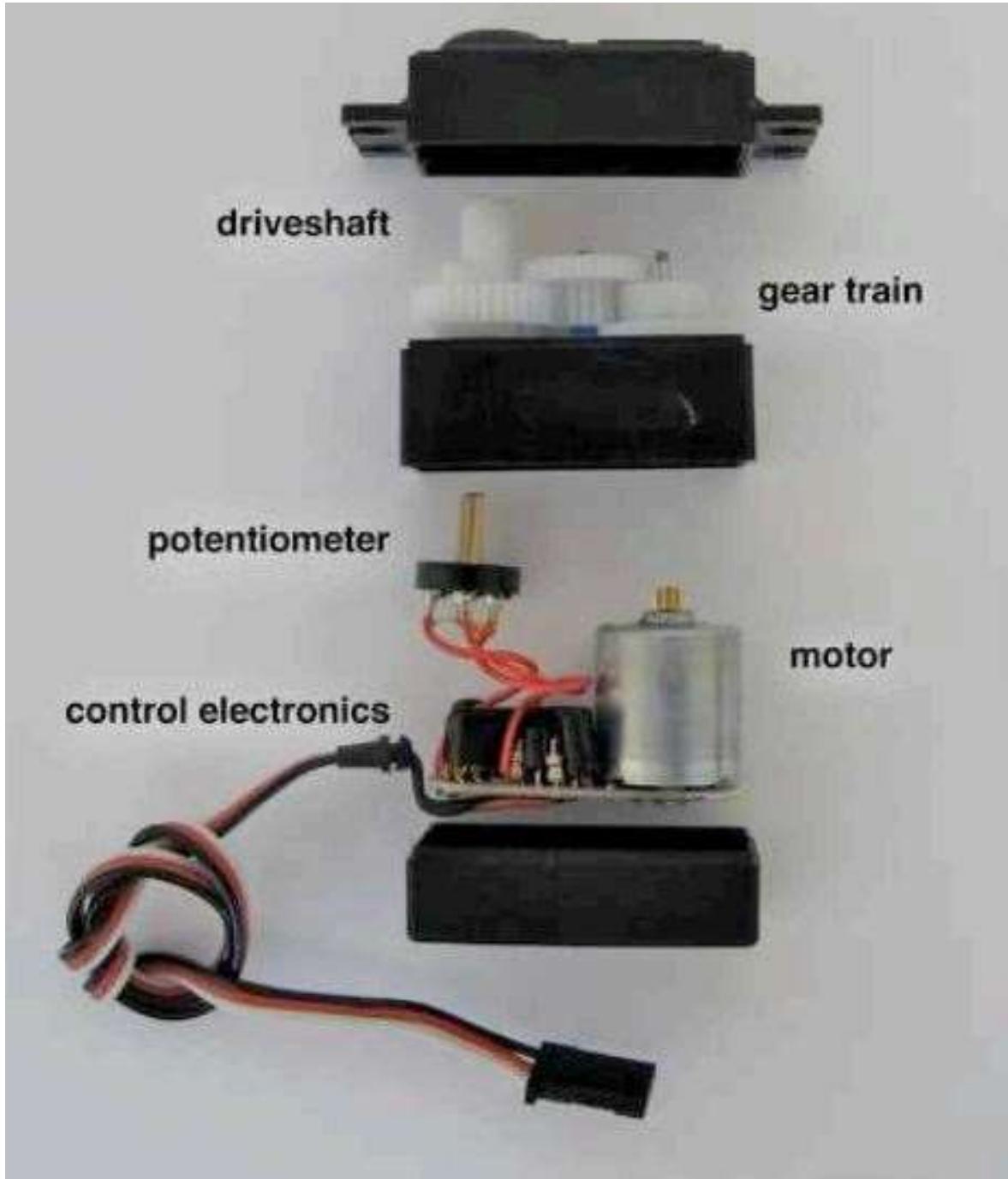


3 WIRES

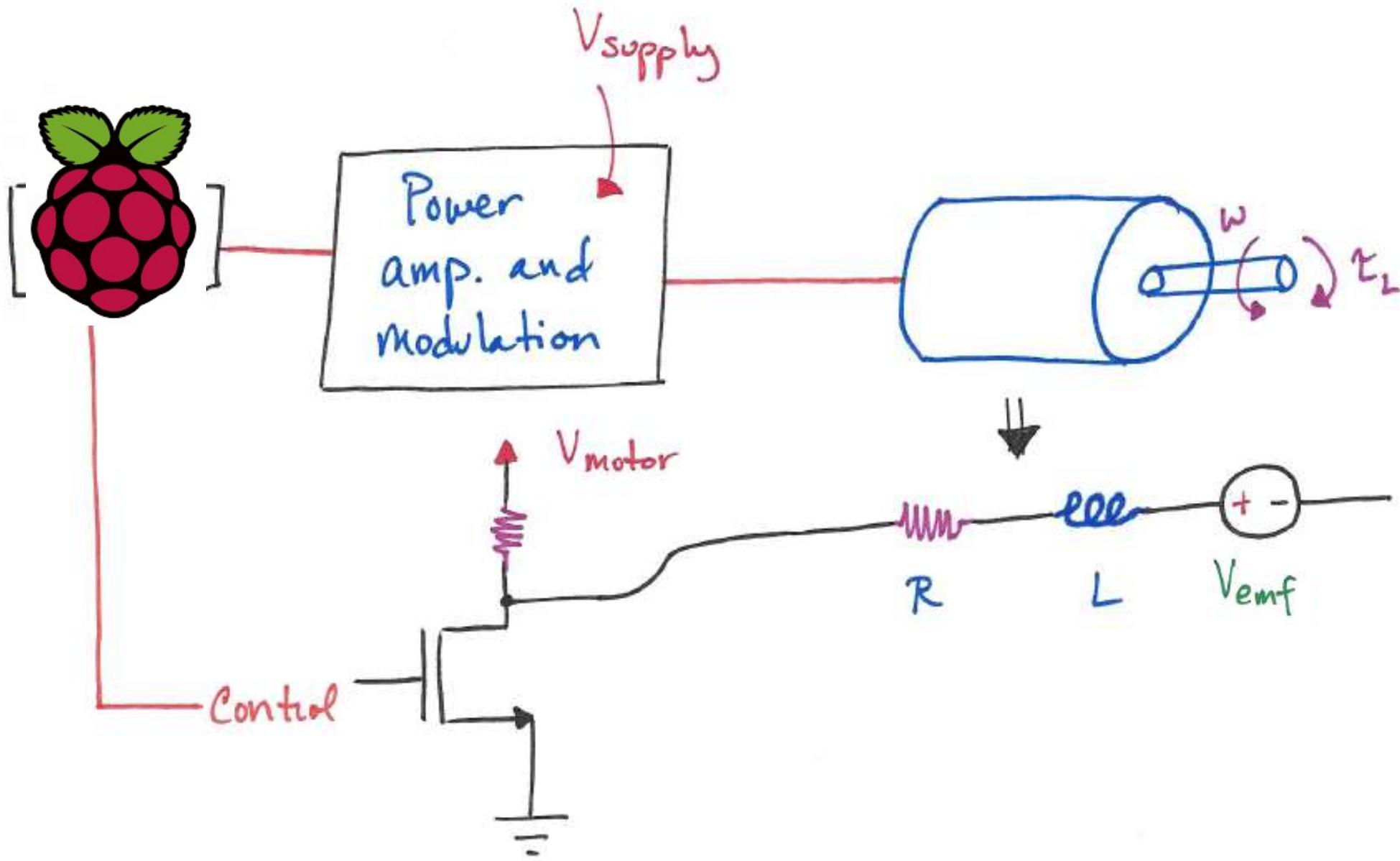
STEPPER MOTOR



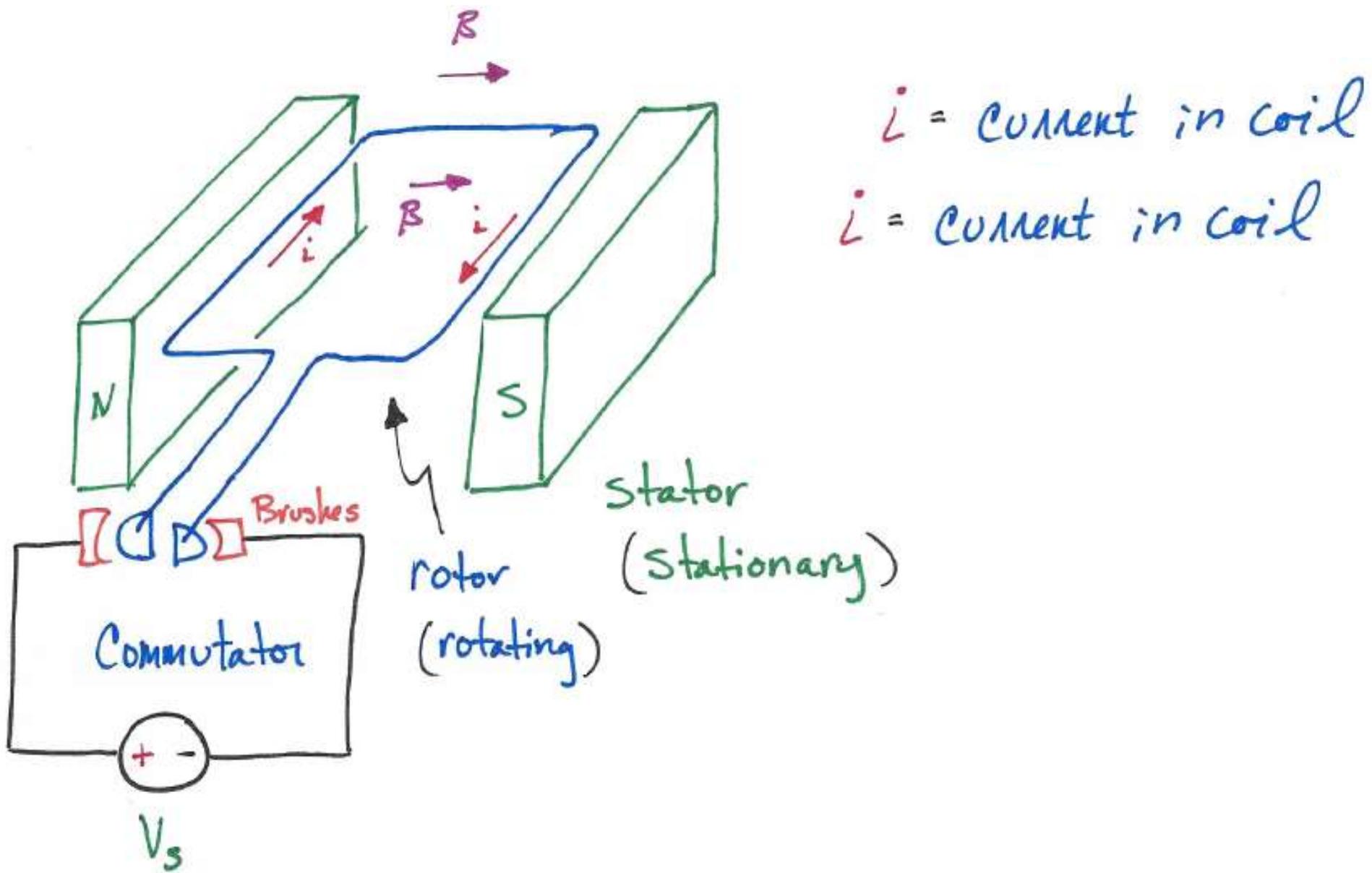
LOTS OF WIRES



Electromagnetic actuation



What's inside a **brushed** DC motor?



What's inside a **brushed** DC motor?

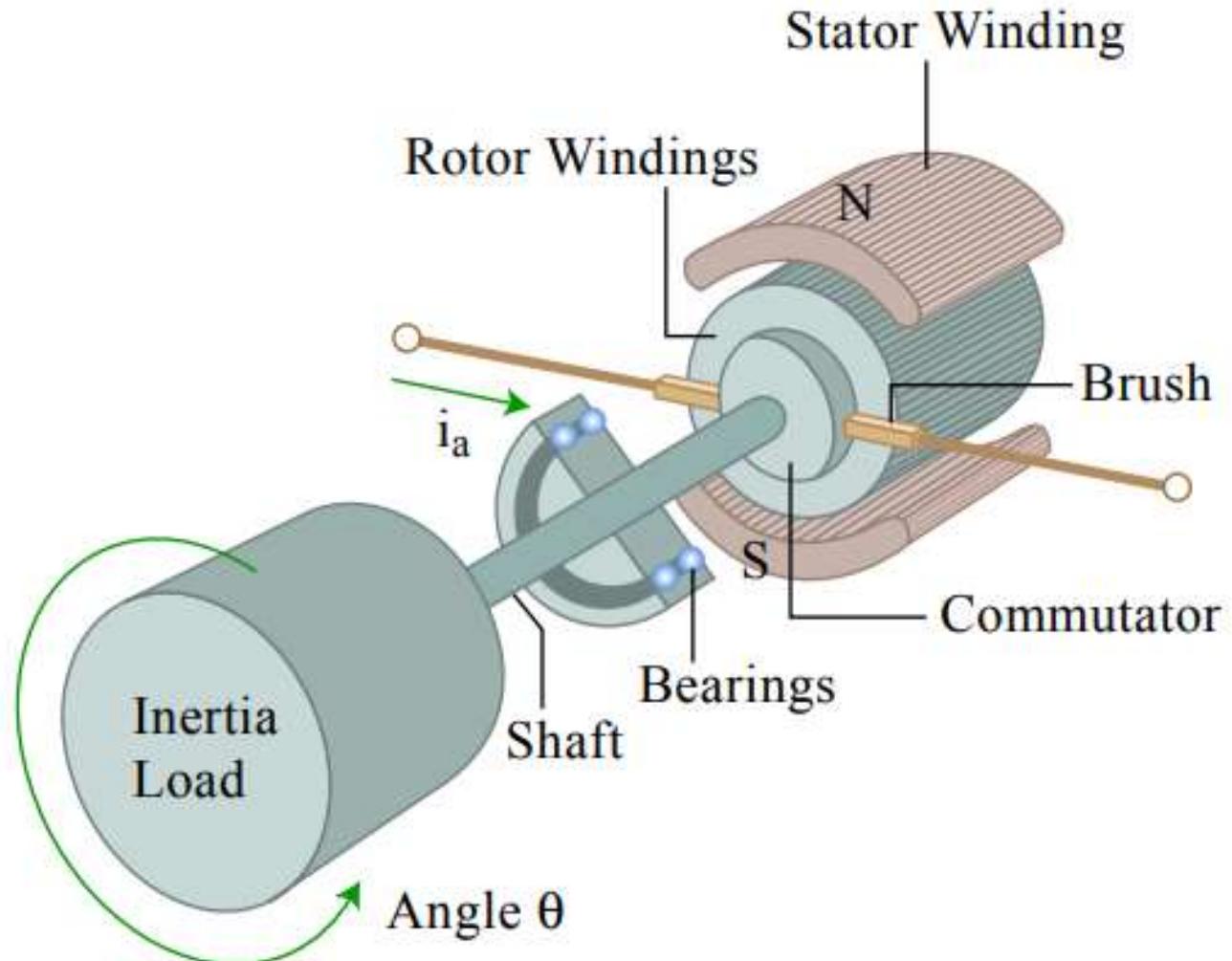


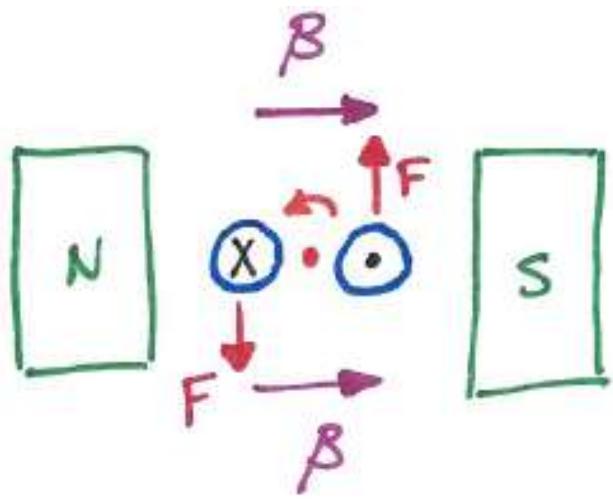
Image: Asada

Motor law

Lorentz Force Law

$$\vec{F} = i \vec{l} \times \vec{\beta}$$

Coil length
Current of coil magnetic field



current

$$F = i l \beta \sin\theta \quad \sin\theta, \theta = 90^\circ$$

$$F = i l \beta$$

motor parameters
torque

$$T = i l_{coil_1} \cdot l_{coil_2} \cdot \beta$$

$$T = K_T i \leftarrow \text{Motor Law}$$

Generator law

- Faraday's Law: rotating a coil through a magnetic field will generate a **voltage** to **oppose** rotation

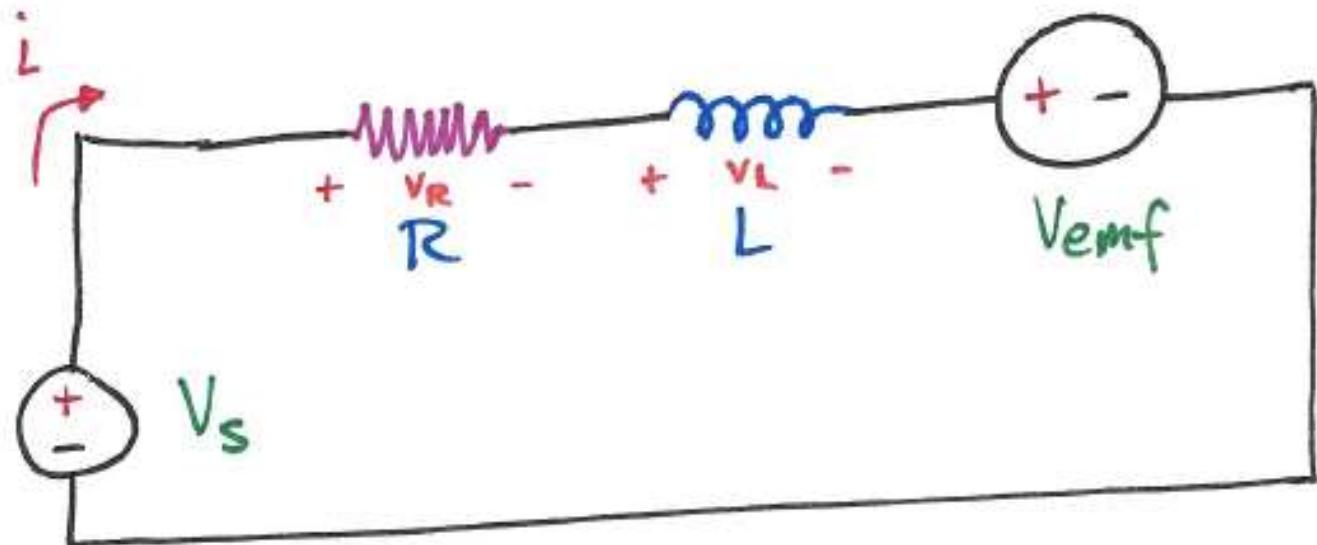
↳ V_{emf} = back emf
↳ electromotive force

$$V_{emf} = K_e \omega$$

speed constant
speed
[rpm, rad/sec]

$$K_e = \text{speed constant}$$
$$\left[\frac{V}{\text{rpm}}, \frac{V}{\text{rad/sec}} \right]$$

DC motor model



$$KVL \rightarrow V_s = V_R + V_L + V_{emf}$$

$$= iR + K_e \omega$$

$$\leftarrow i = \frac{T}{K_T} \quad \uparrow \omega = \text{speed}$$

$$\therefore V_s = \frac{T}{K_T} R + K_e \omega$$

R = coil resistance [Ω]

L = coil inductance [H]

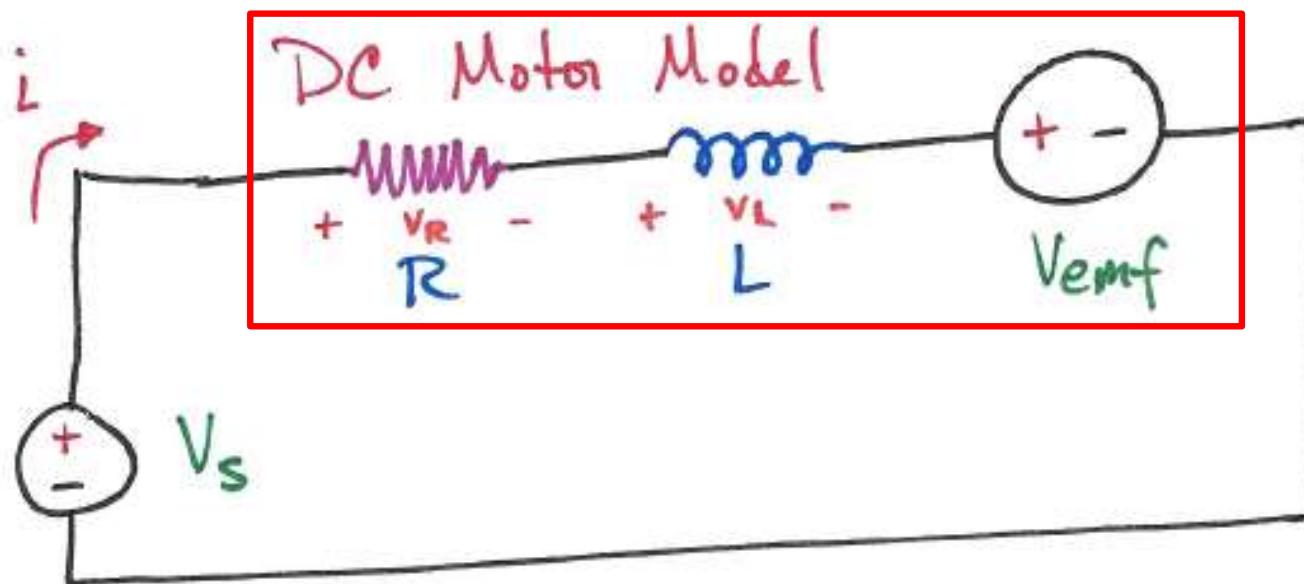
$V_{emf} = K_e \omega$

V_s = supply voltage

i = coil current

$$V_L = L \frac{di}{dt}$$

DC motor model



$$\boxed{\omega} = \frac{V_s}{K_e} - \frac{R}{K_e K_T} \cdot \boxed{T}$$

$$\boxed{T} = \frac{K_T V_s}{R} - \frac{K_e K_T}{R} \cdot \boxed{\omega}$$

$$V_s = \frac{T}{K_T} R + K_e \omega$$

torque constant

resistance in coil

speed

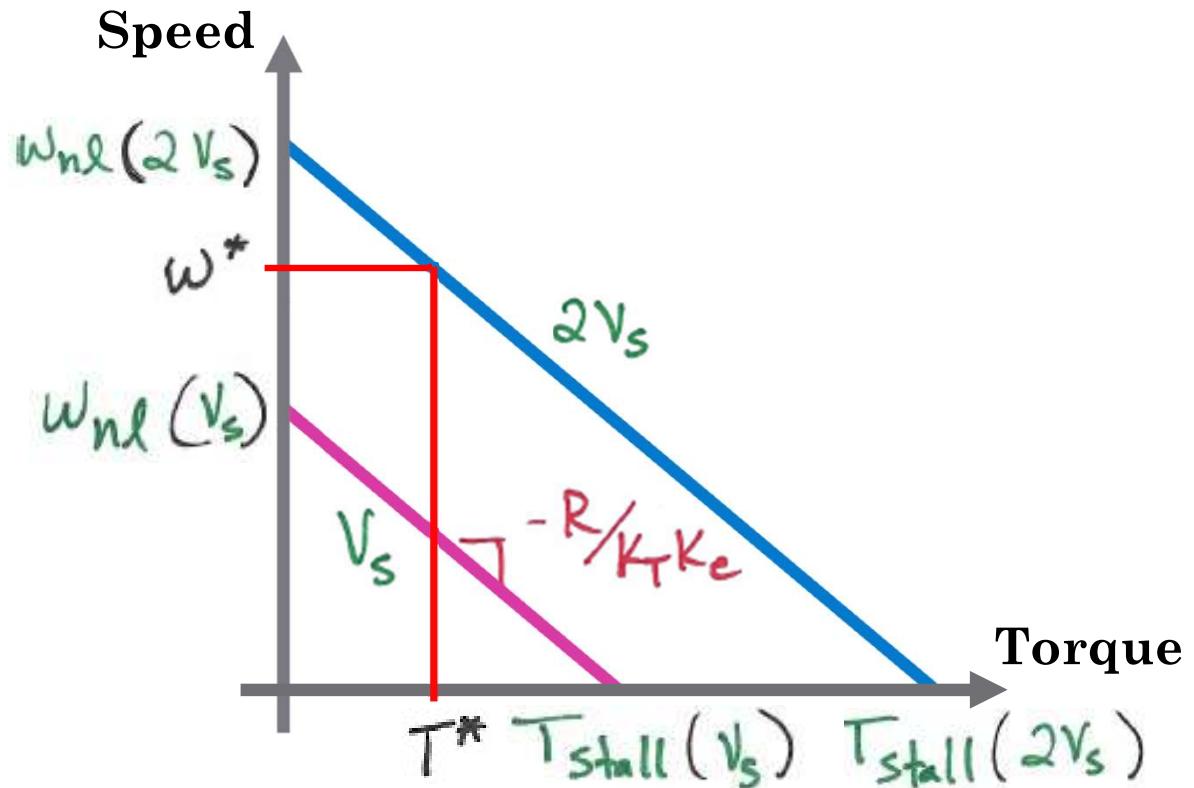
Torque vs. speed

$$\omega = \frac{V_s}{K_e} - \frac{R}{K_e K_T} \cdot T$$

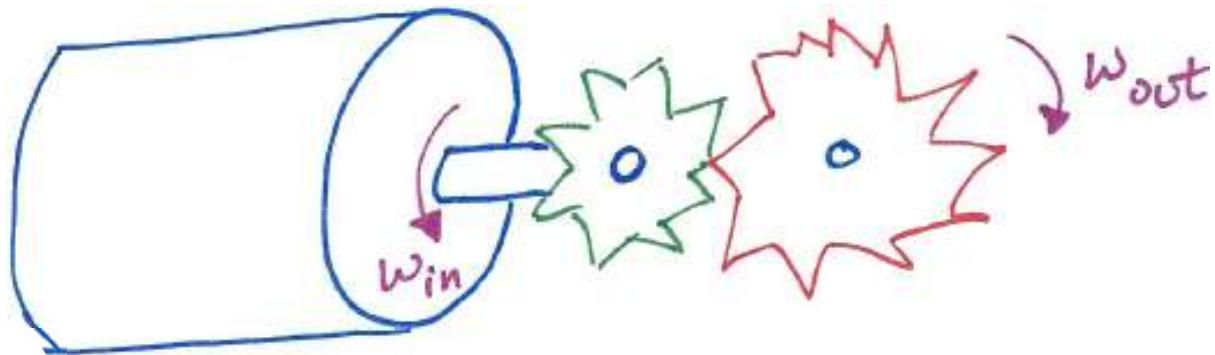
$$\omega_{\text{no-load}} = \frac{V_s}{K_e} = \omega_{\text{max}}$$

$$T = \frac{K_T V_s}{R} - \frac{K_e K_T}{R} \cdot \omega$$

$$T_{\text{stall}} = \frac{K_T V_s}{R} = T_{\text{max}}$$



DC motor gearing

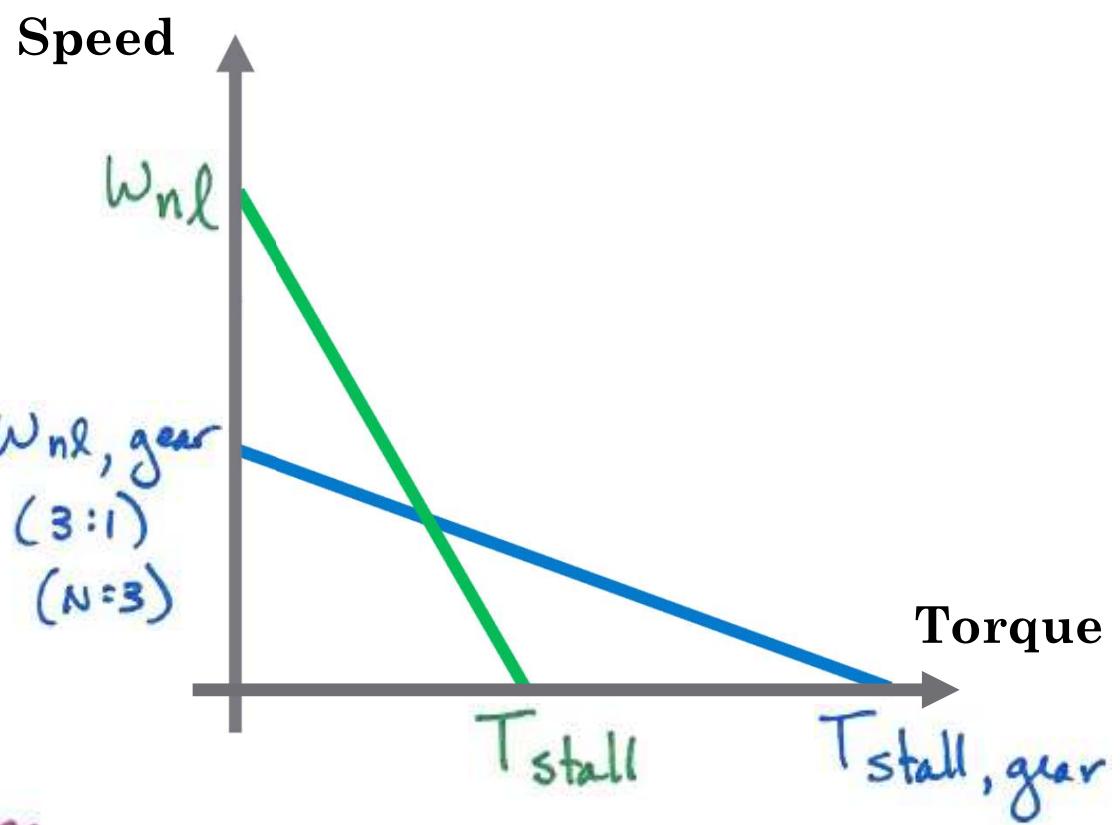


bear Ratio

$$N = \frac{w_{in}}{w_{out}} = \frac{\text{input speed from motor}}{\text{output speed from gearbox}}$$

$$w_{nl} = 1000 \text{ rpm}$$

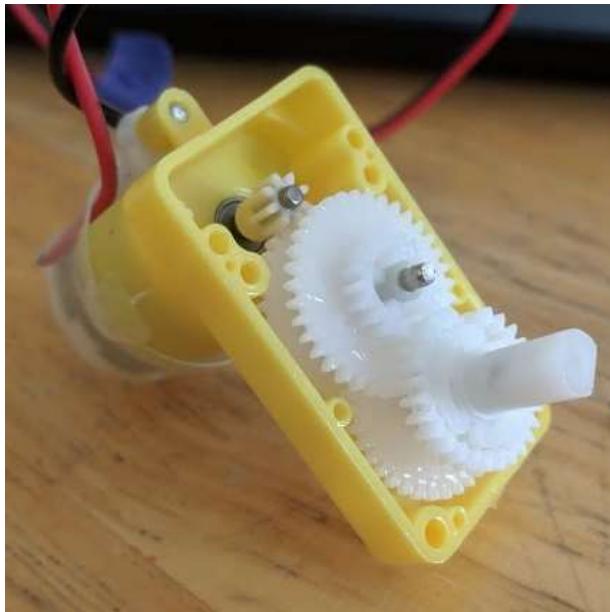
10:1 gear ratio ($N=10$)



$$h_{\text{gearbox}} = \frac{T_{out} w_{out}}{T_{in} w_{in}}$$

$$\frac{T_{out}}{T_{in}} = N \cdot h_{\text{gearbox}}$$

DC motor gearing



Motor Feature :

- Gear Ratio : 1:120
- No-load speed (3V): 100RPM
- No-load speed (6V): 200RPM
- No-load current (3V): 60mA
- No-load current (6V): 71mA
- Stall current (3V): 260mA
- Stall current (6V): 470mA
- Torque (3V): 1.2Kgcm
- Torque (6V): 1.92Kgcm
- Weight : 45g

Assembly

- Assemble encoders onto motors



Assembly

* *Use small zip ties* *

- Assemble encoders onto motors



<https://www.youtube.com/watch?v=oLBYHbLO8W0>

Assembly

- Assemble encoders onto motors

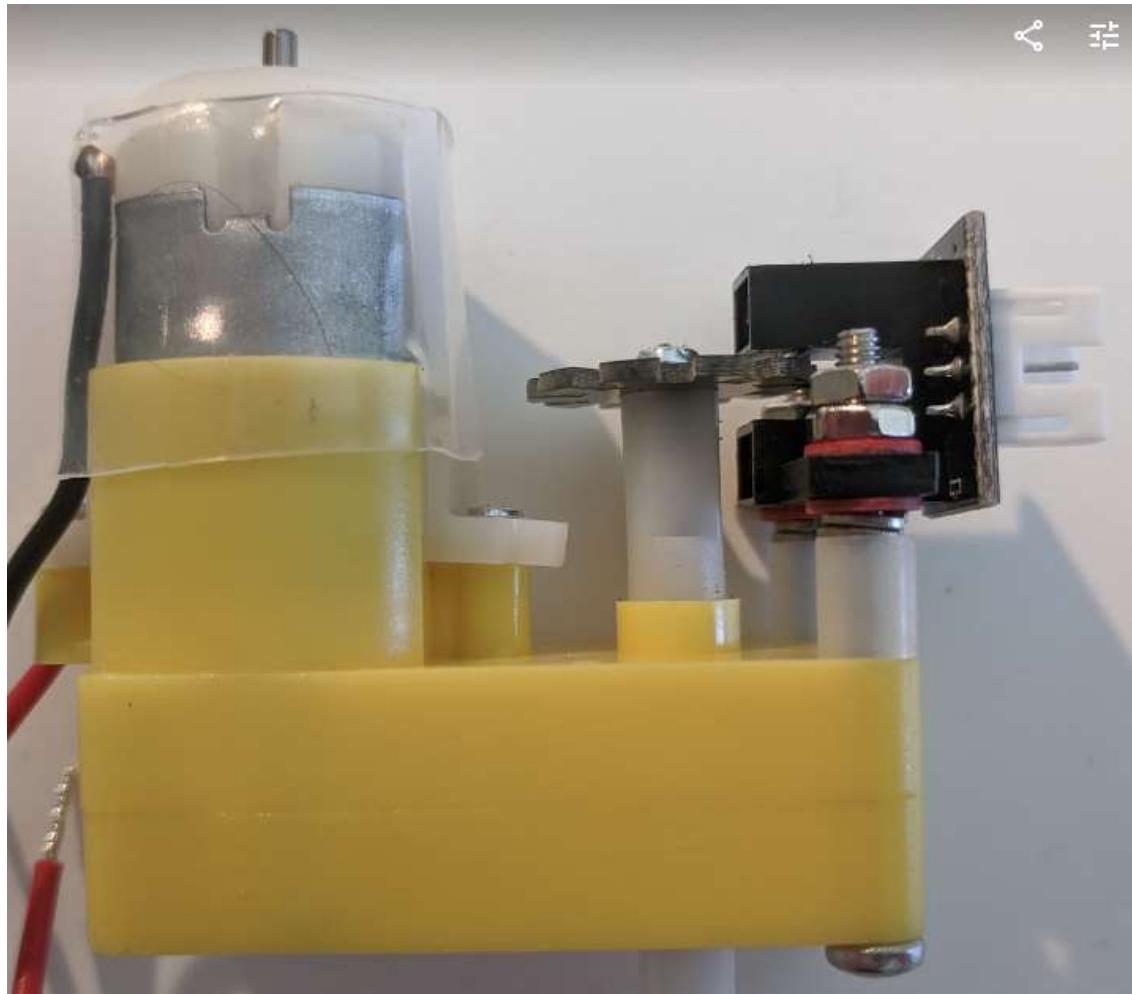


<https://www.youtube.com/watch?v=oLBYHbLO8W0>

Assembly

Mount motors to vehicle chassis first

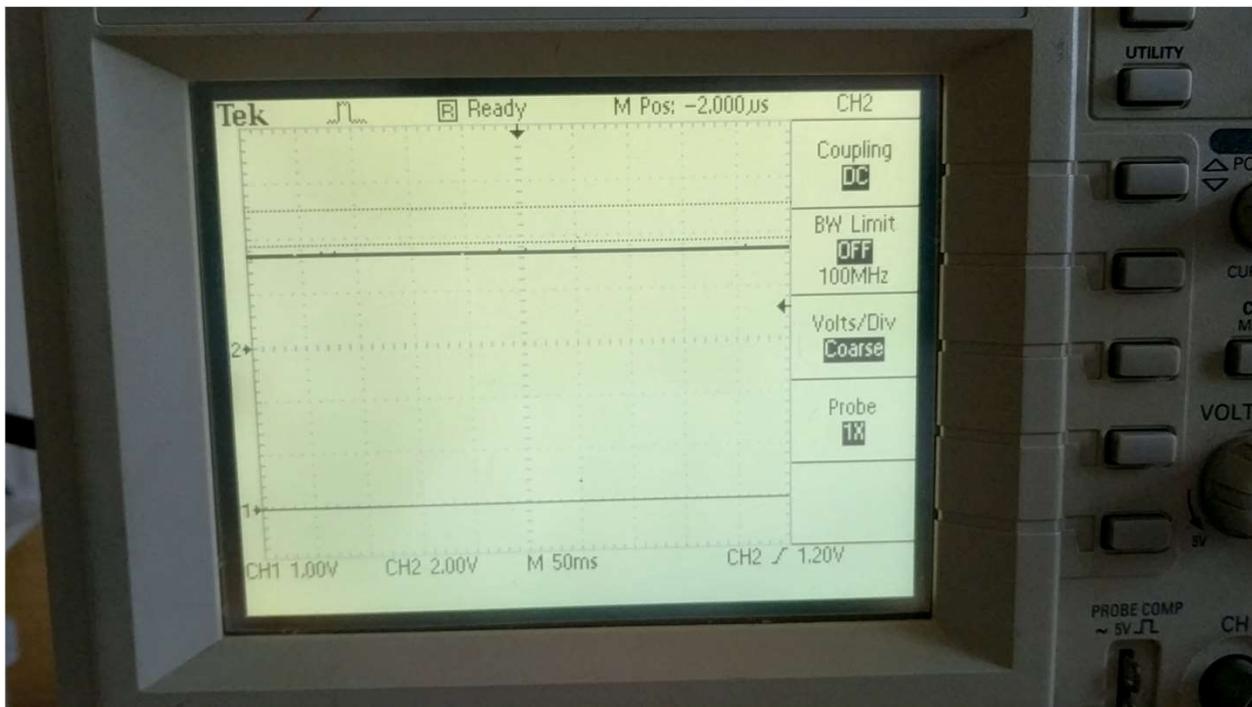
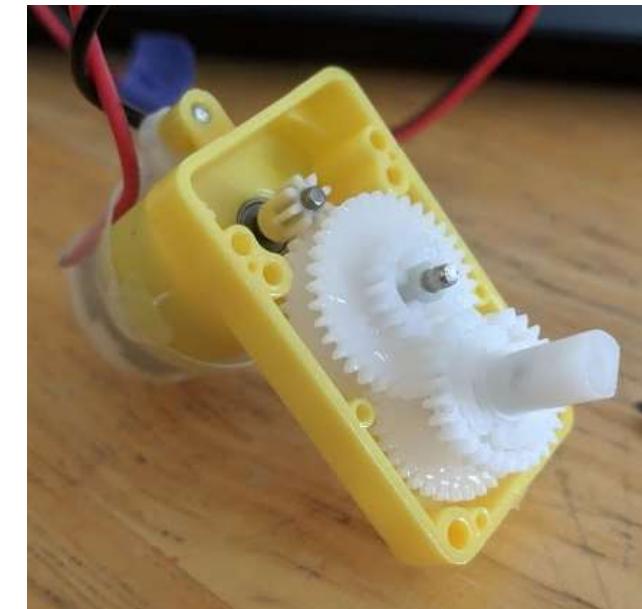
- Assemble encoders onto motors



Encoders

- Electro-mechanical device
- Converts motion into sequence of digital pulses
- Pulse train converted to position measurements
 - Robot localization
- Typically magnetic or optical

Proprioceptive Sensor



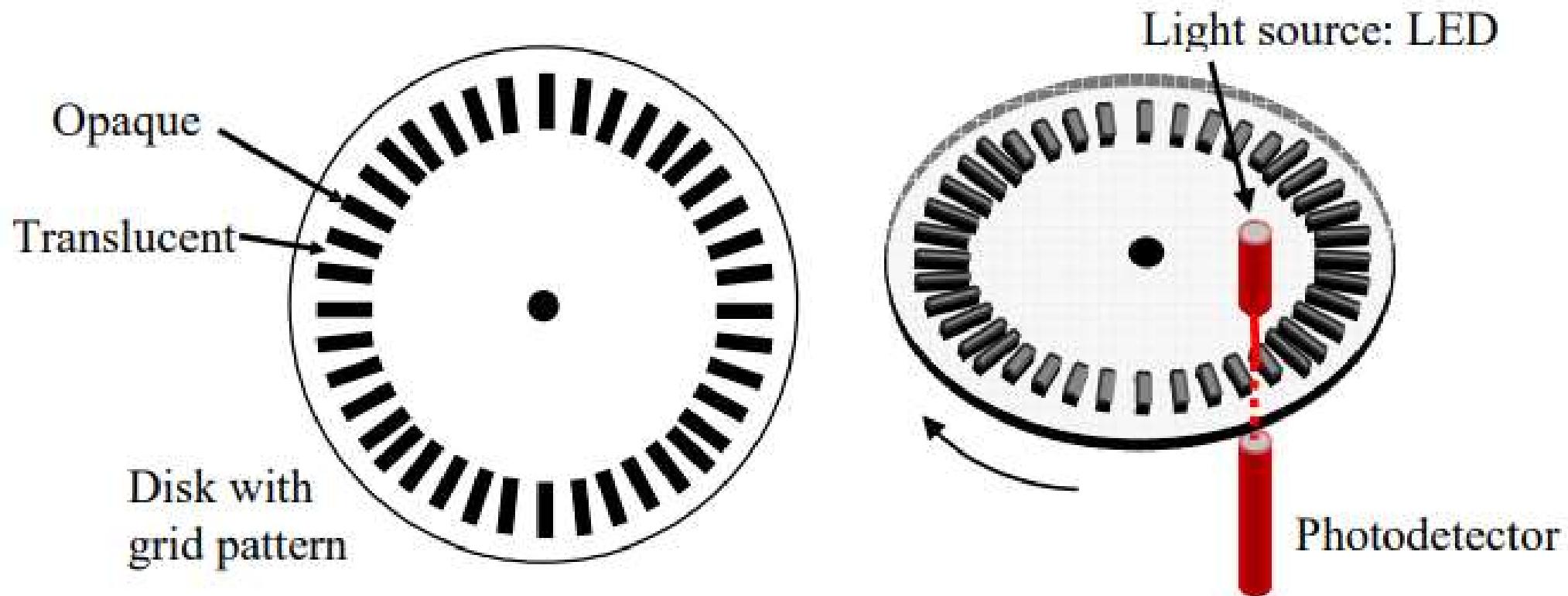


Figure 2.5.1 Basic construction of optical shaft encoder

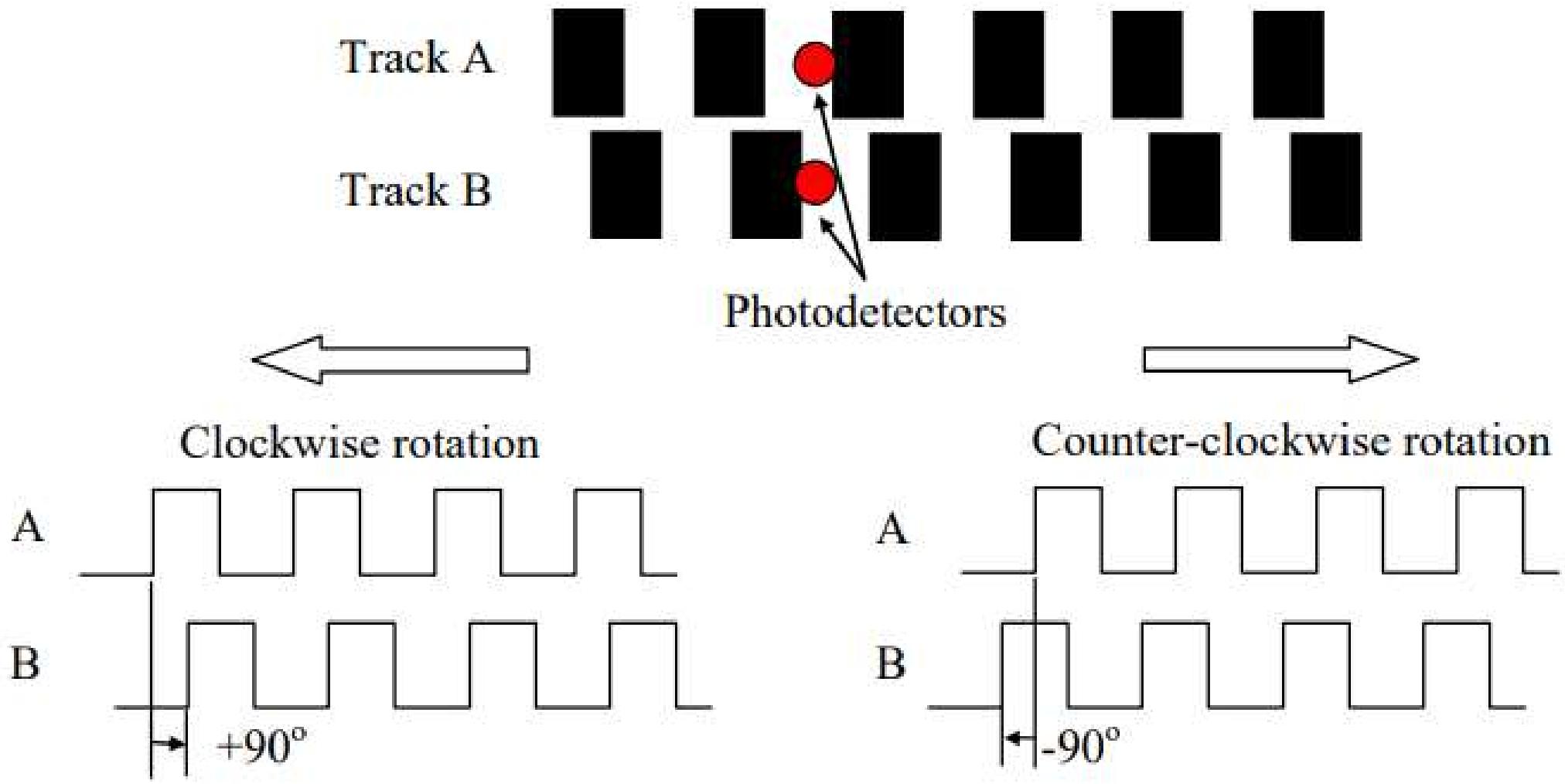
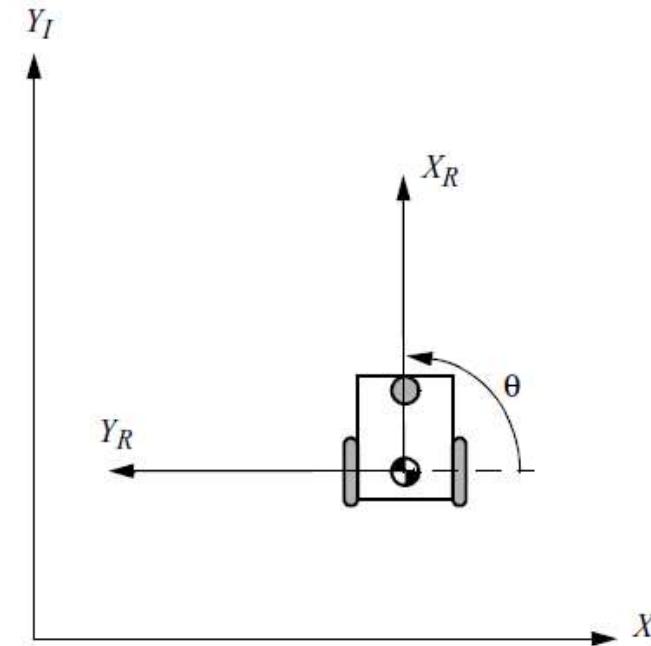
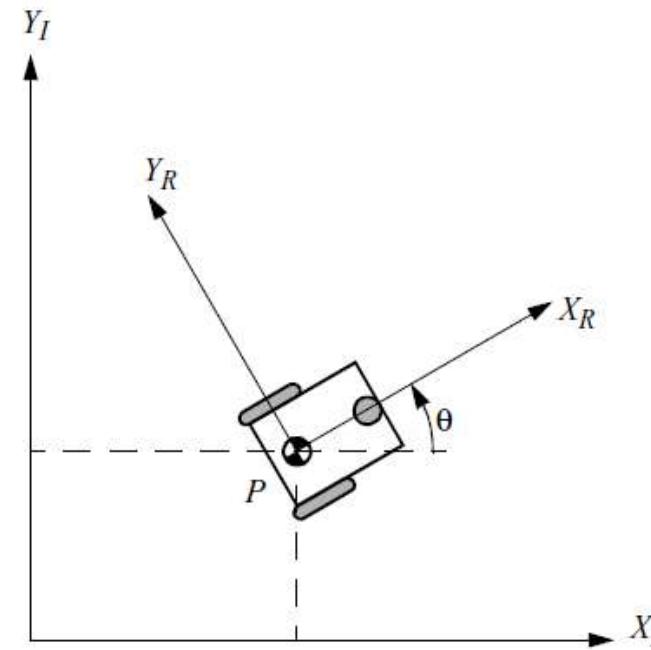


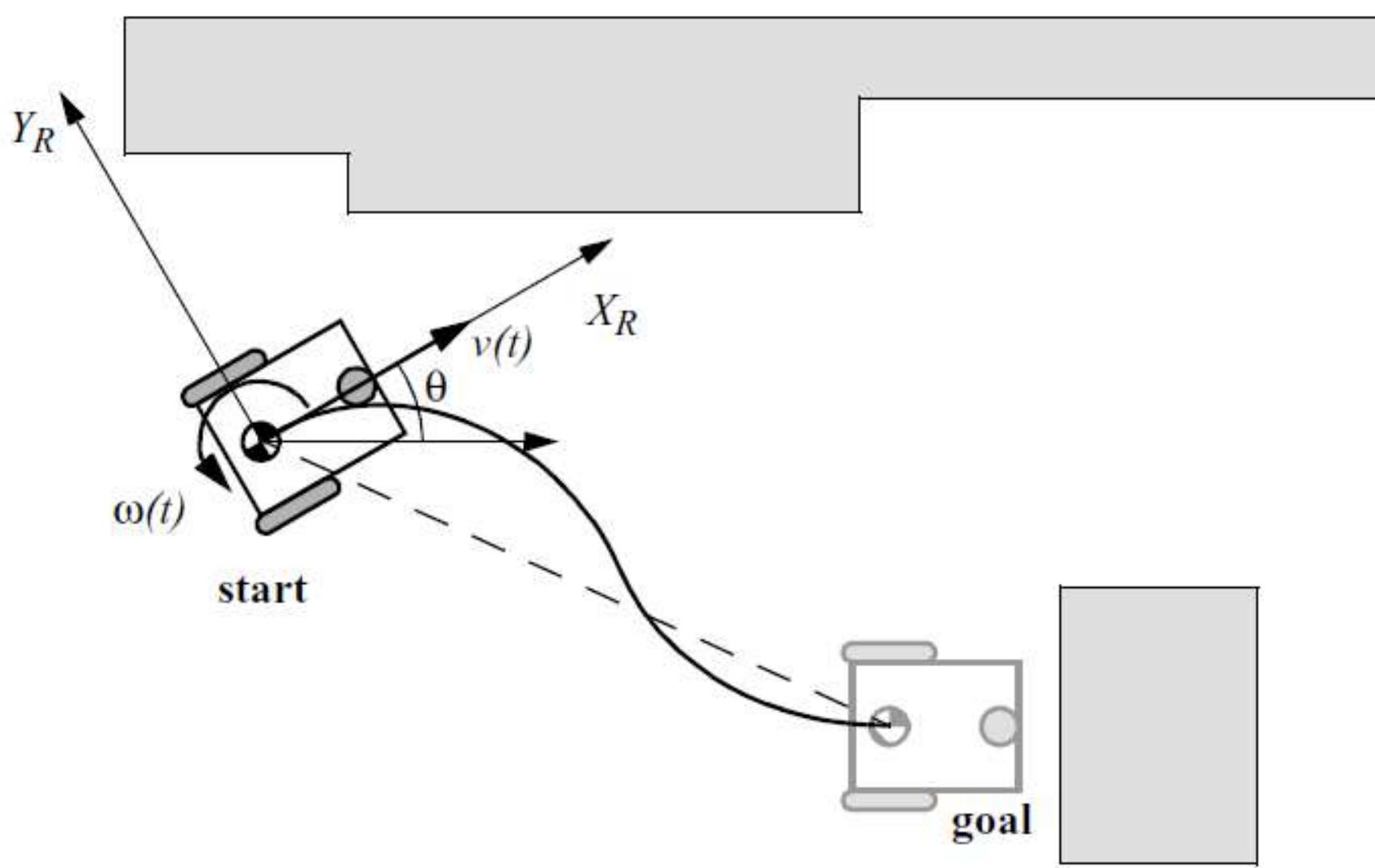
Figure 2.5.2 Double track encode for detection of the direction of rotation

Kinematics

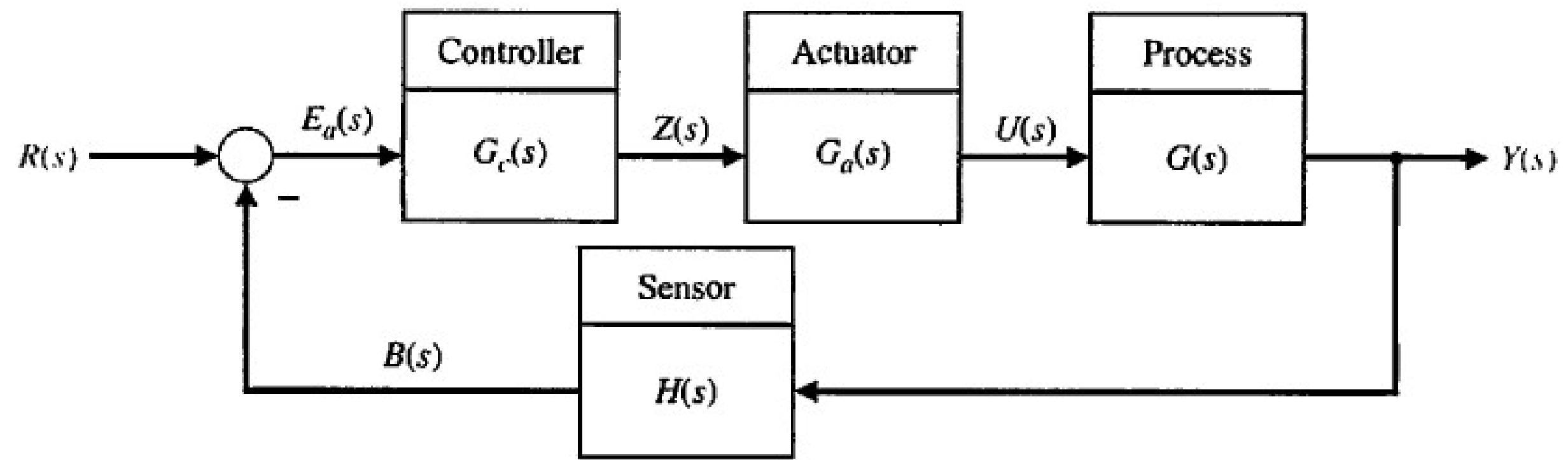
- Study of how mechanical systems behave
- Mobile robotics
 - Design for specific tasks
 - Create control software to drive hardware
- Global reference frame & robot local reference frame



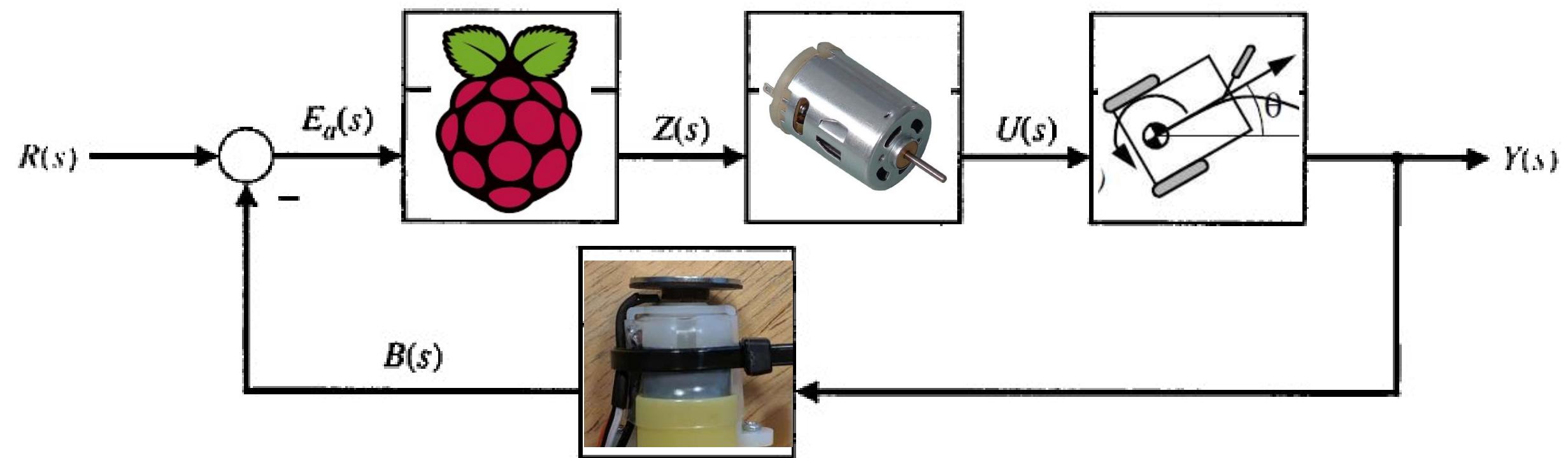
Kinematics



Control Theory

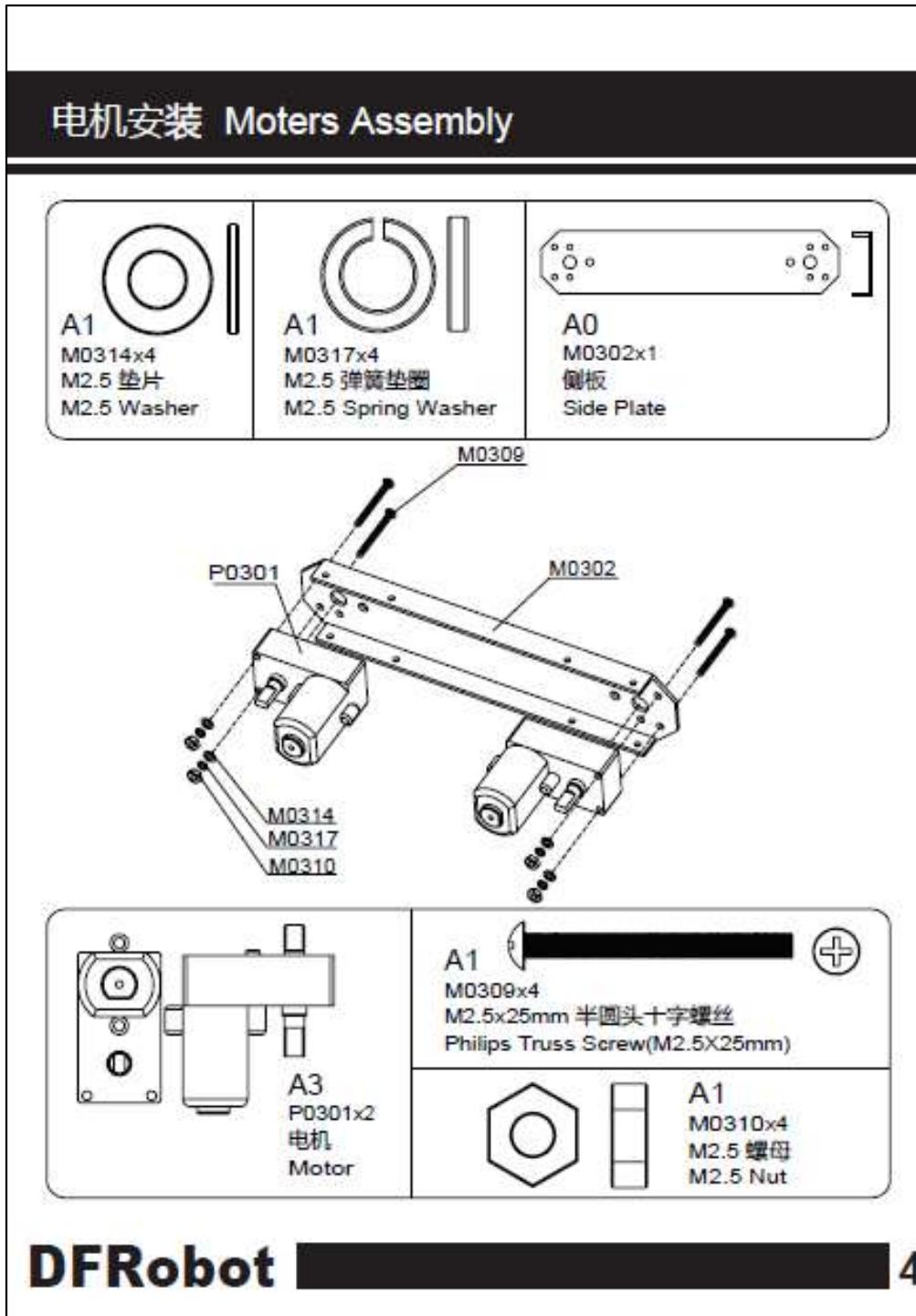


Control Theory



Assembly

- Assemble motors to side plates



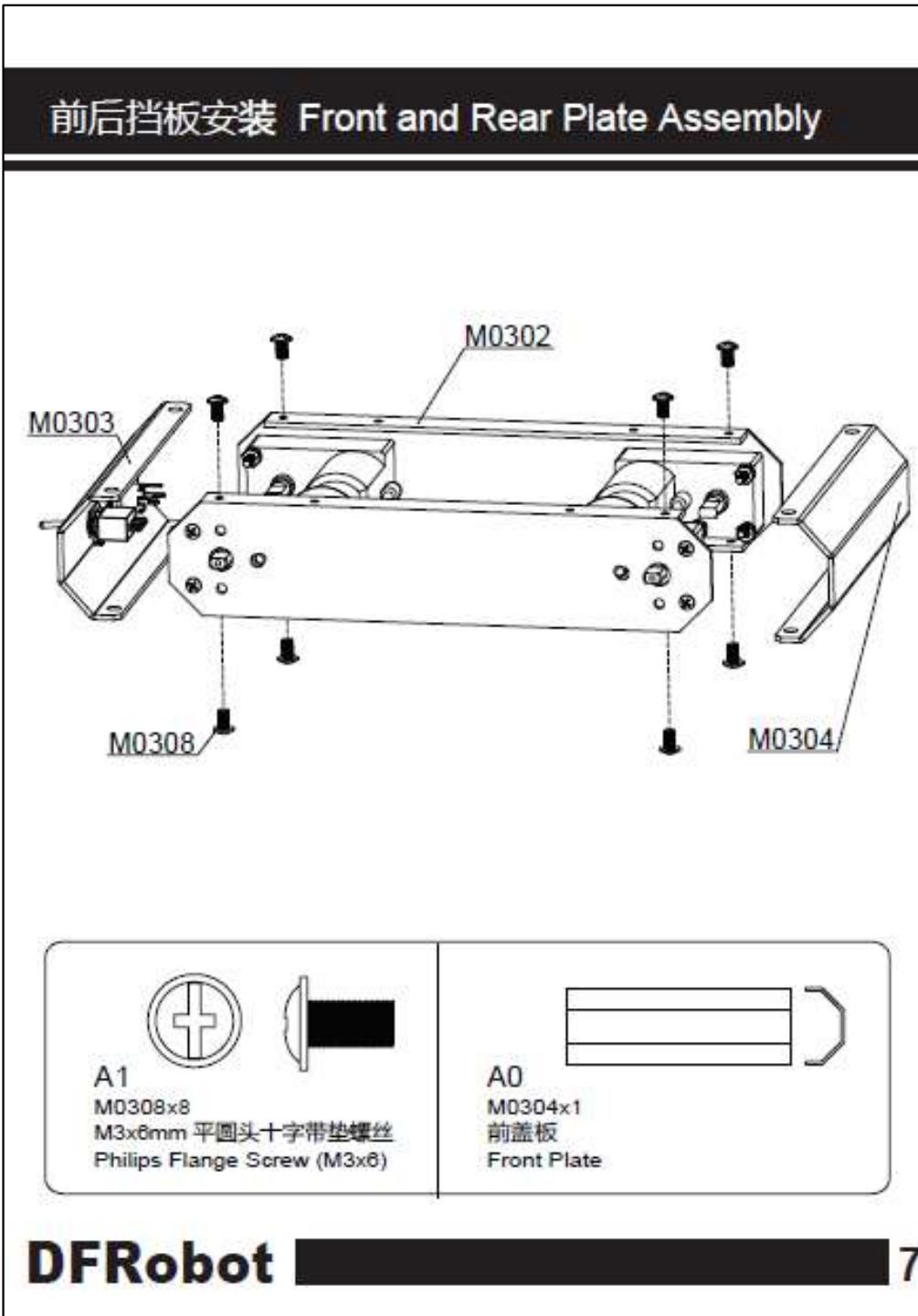
Assembly

- Assemble motors to side plates



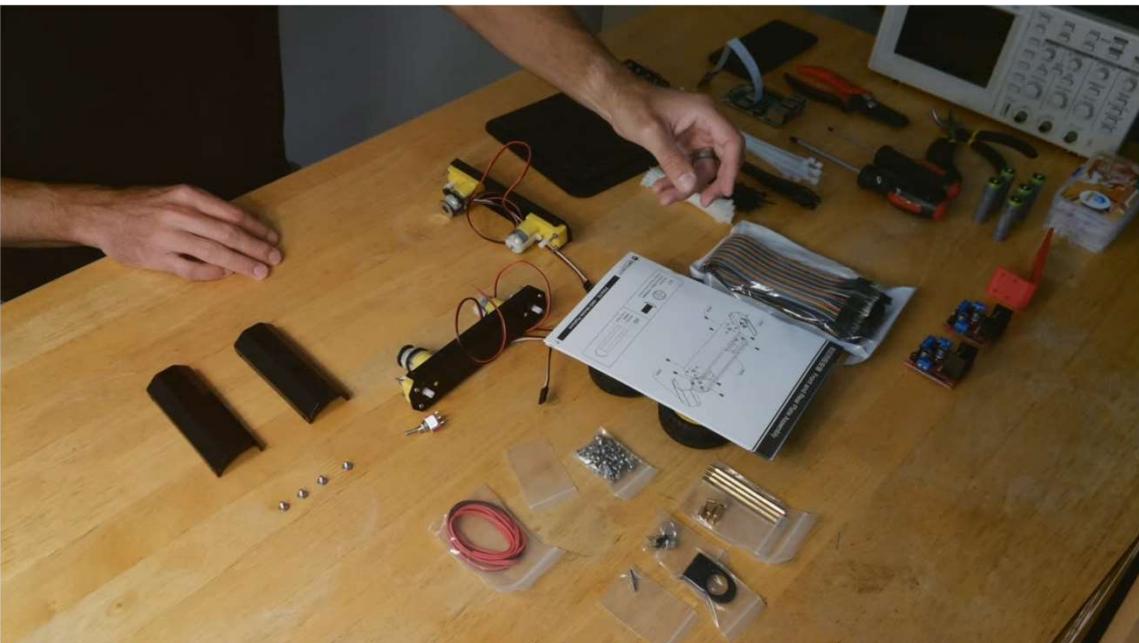
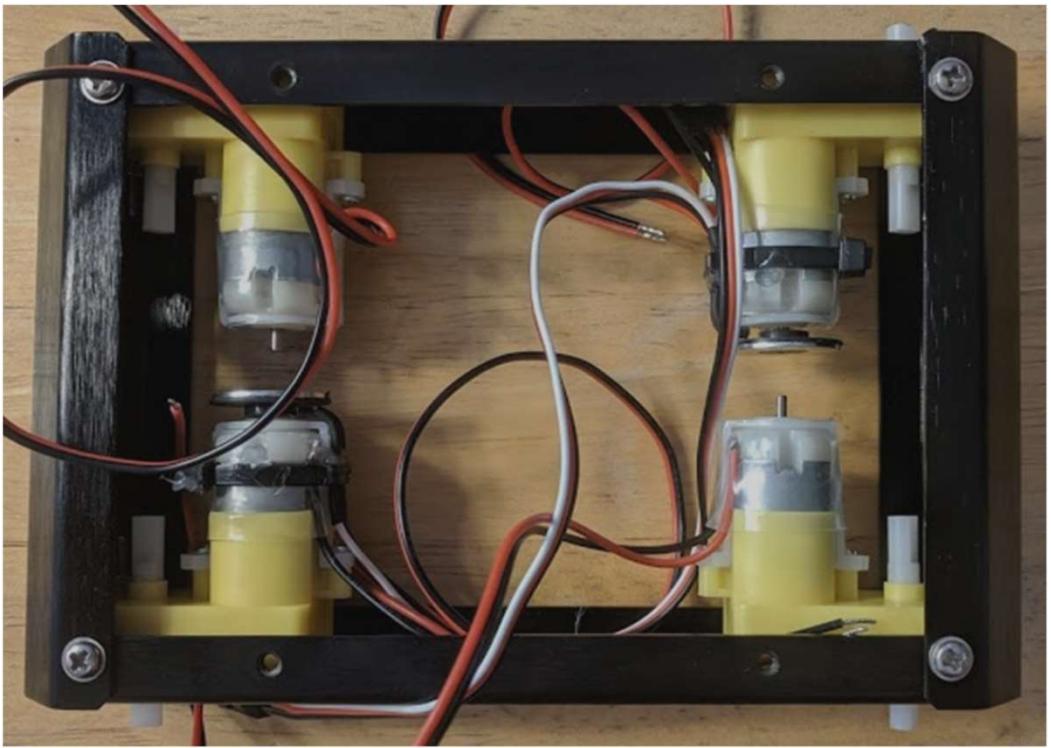
Assembly

- Assemble front & rear plates



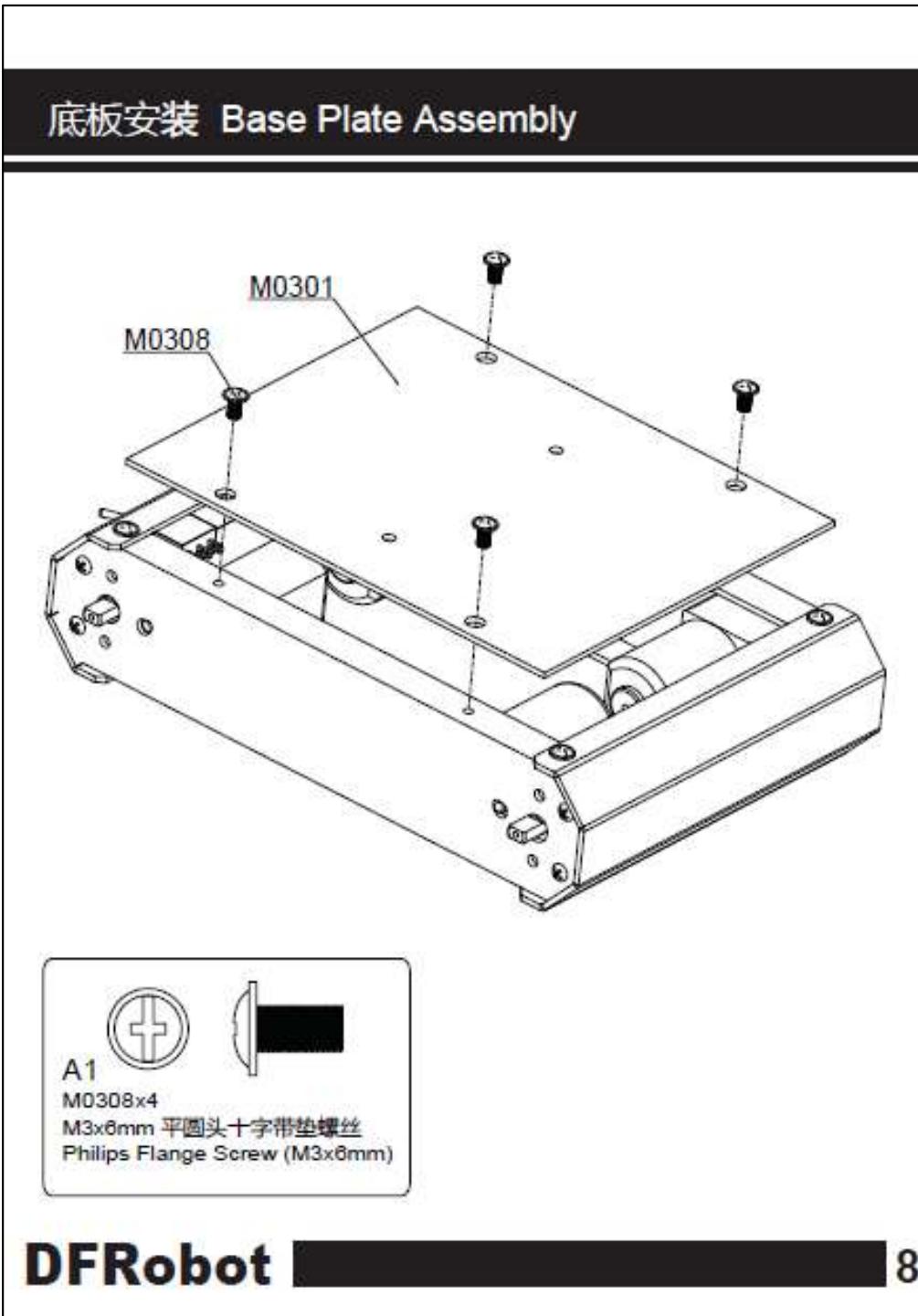
Assembly

- Assemble front & rear plates



Assembly

- Assemble base plate to frame



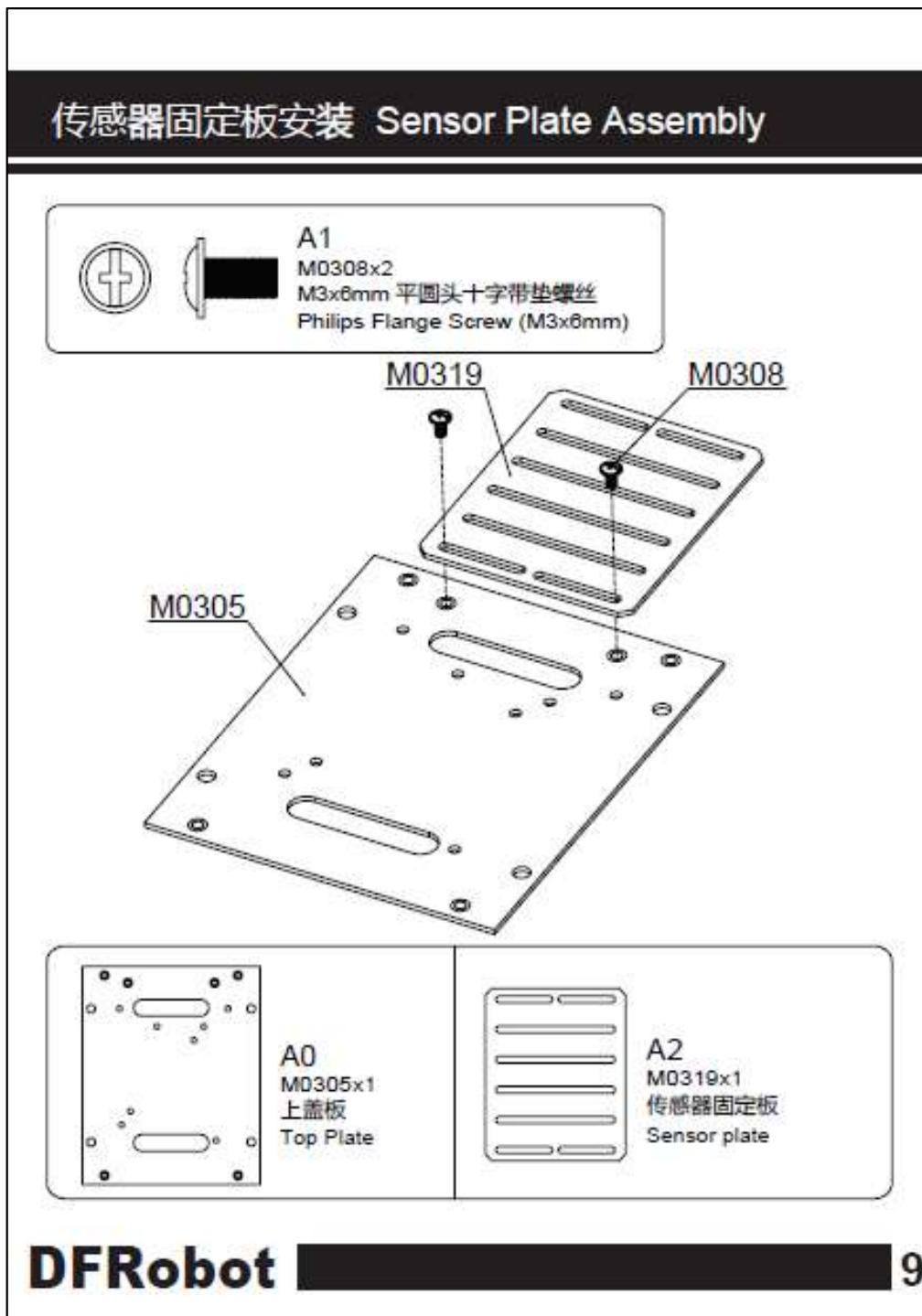
Assembly

- Assemble base plate to frame



Assembly

- Assemble sensor plate to top plate



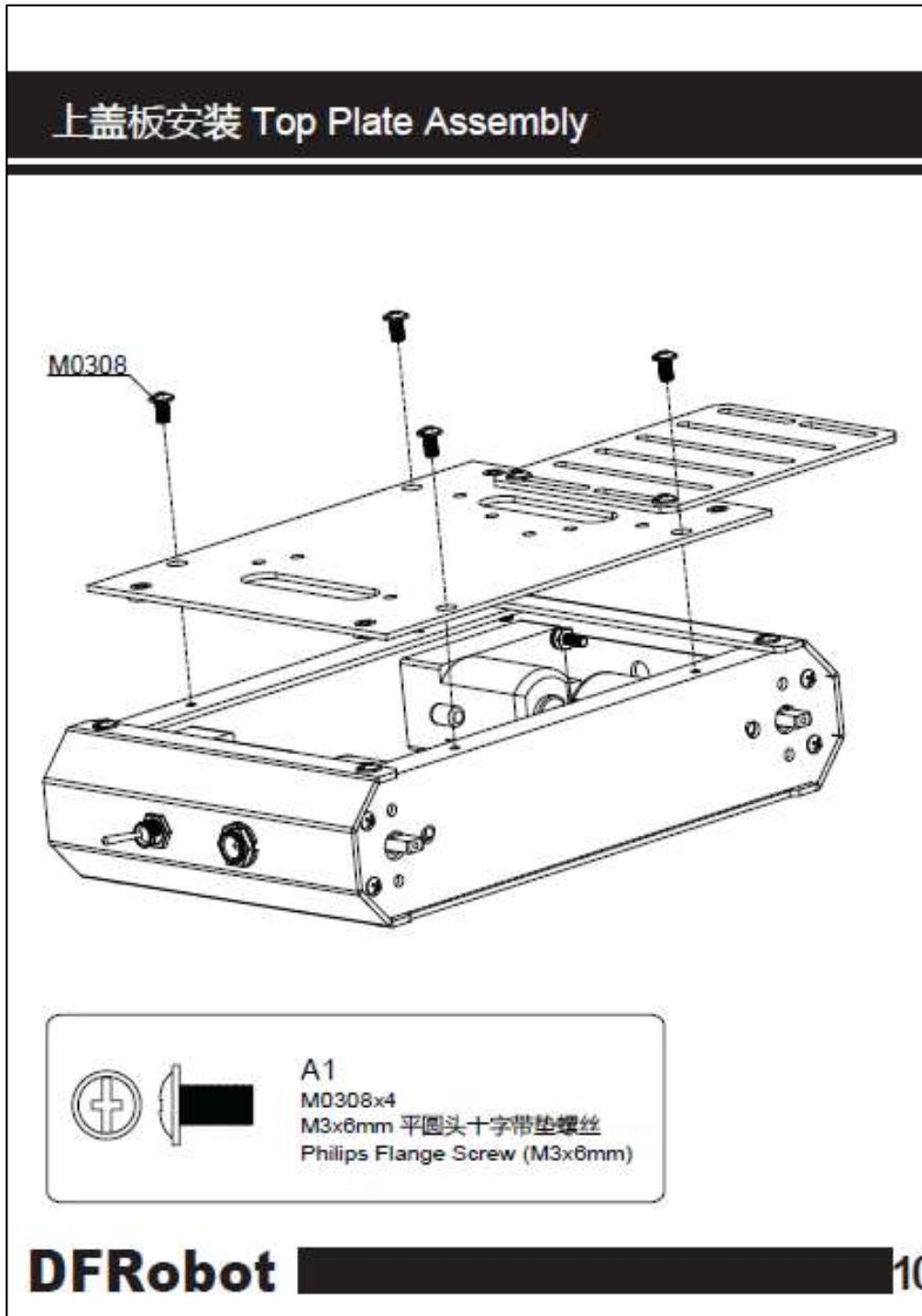
Assembly

- Assemble sensor plate to top plate



Assembly

- Assemble top plate to frame



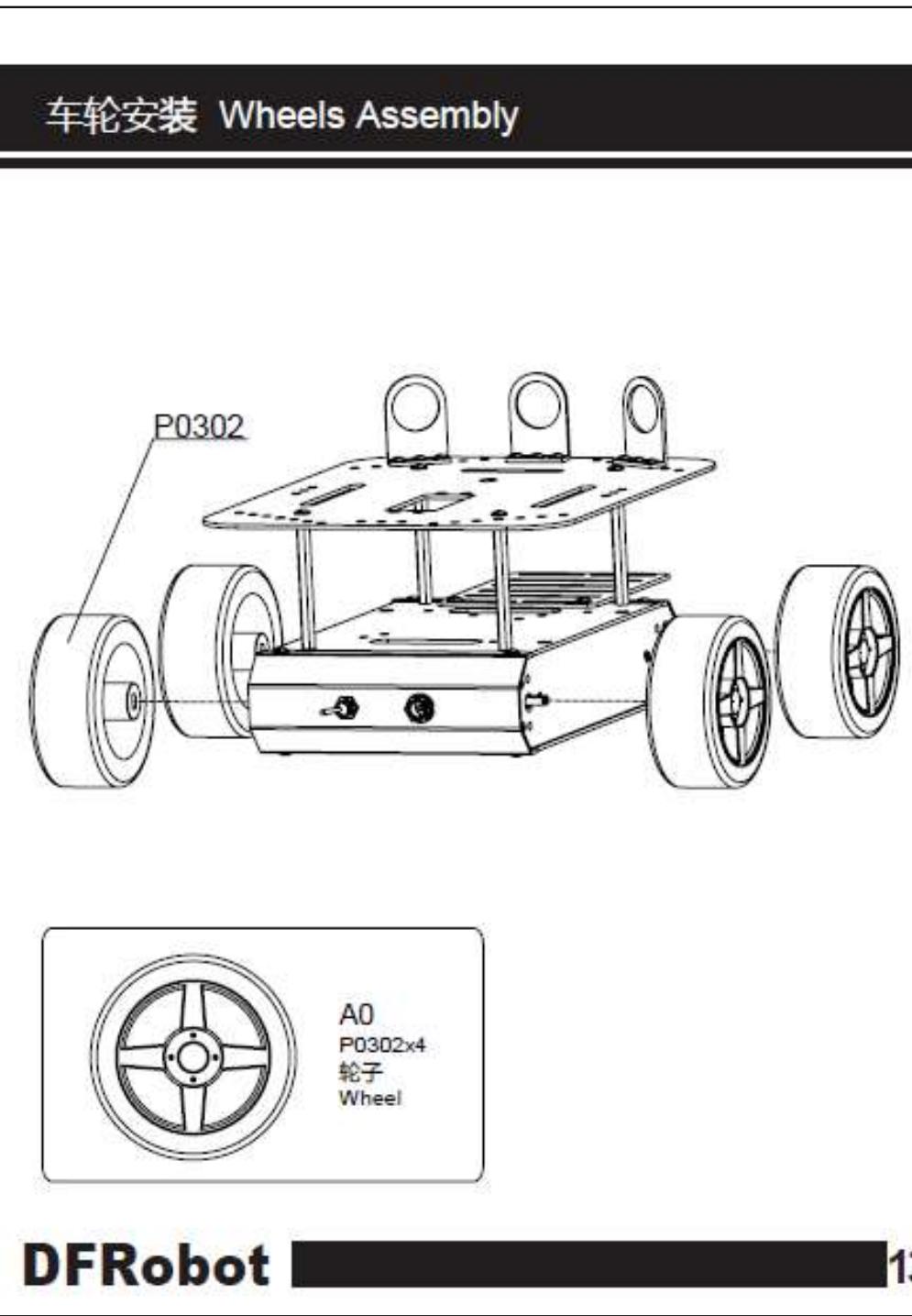
Assembly

- Assemble top plate to frame



Assembly

- Mount wheels onto motors



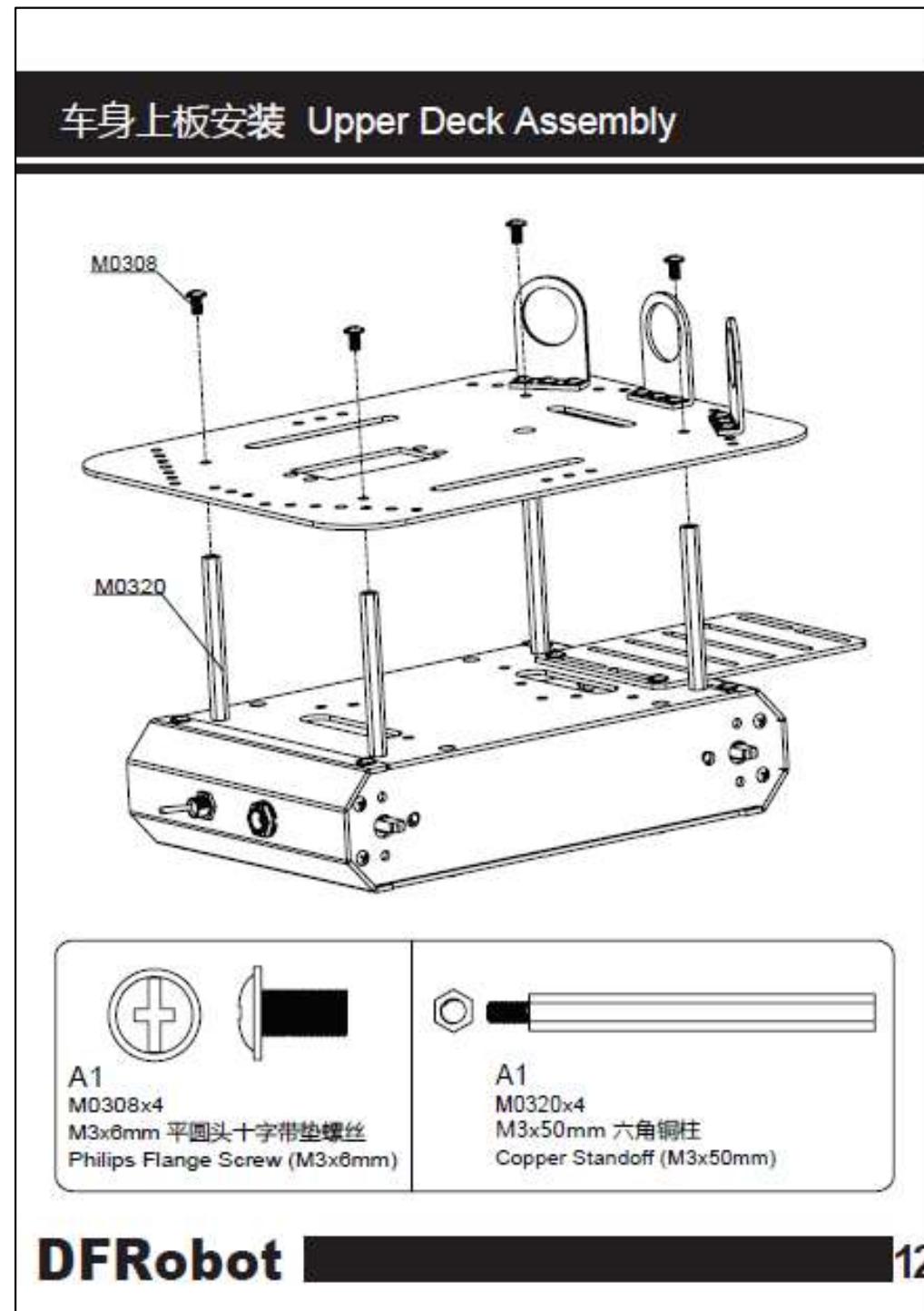
Assembly

- Mount wheels onto motors



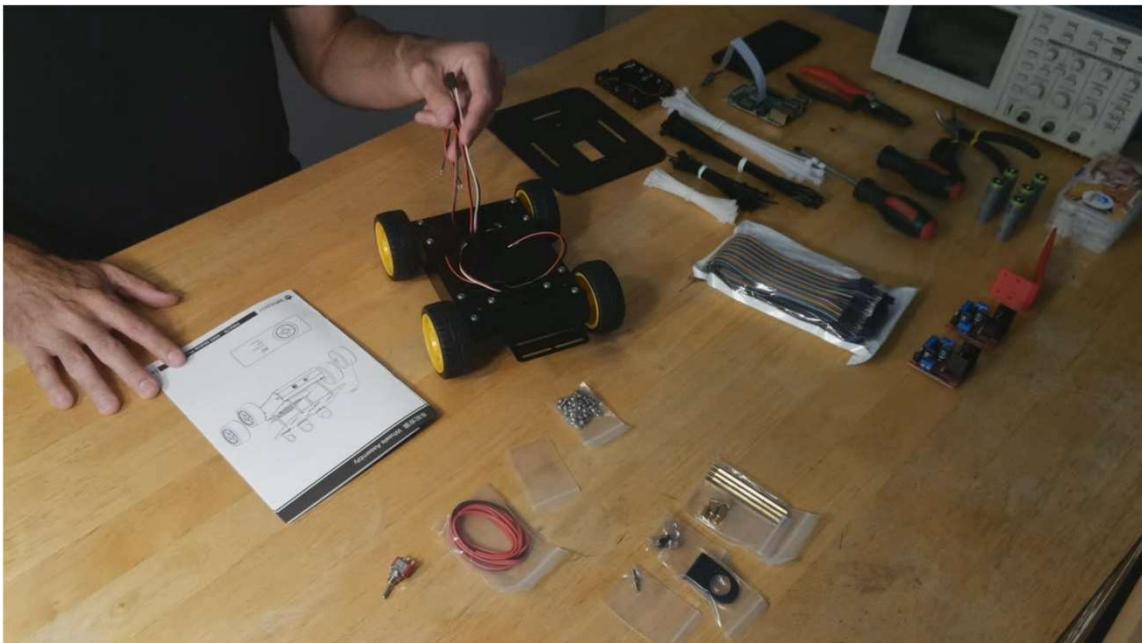
Assembly

- Assemble standoffs for upper deck



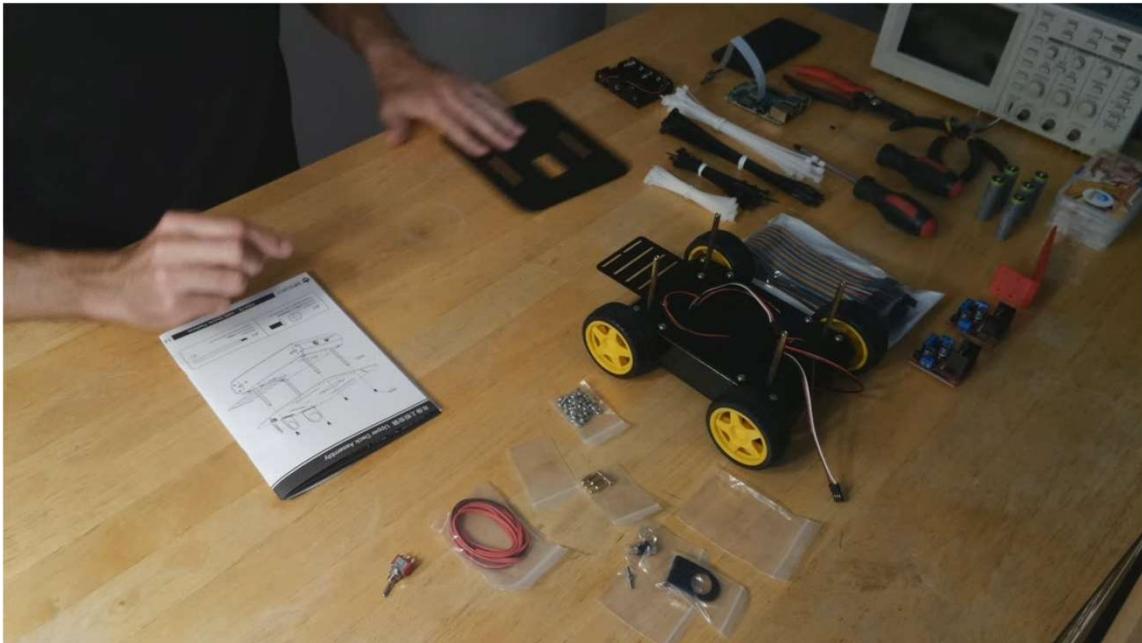
Assembly

- Assemble standoffs for upper deck



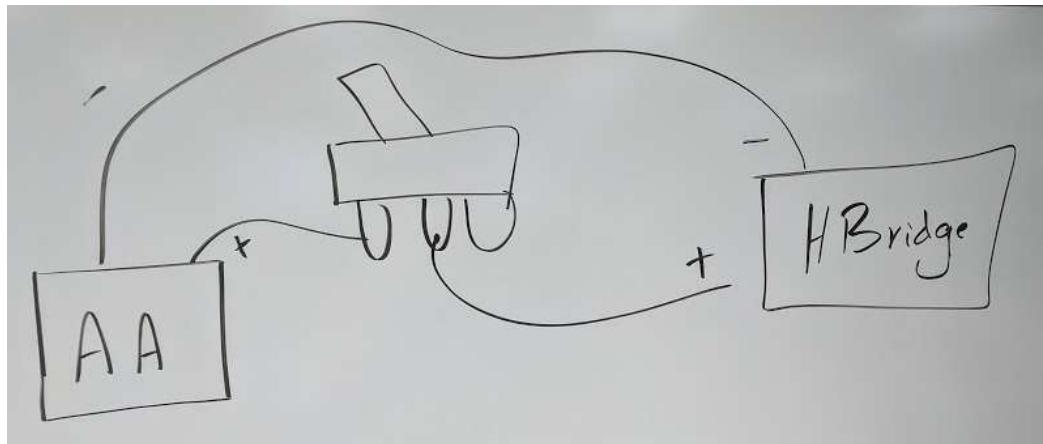
Assembly

- Mount upper deck on standoffs



Assembly

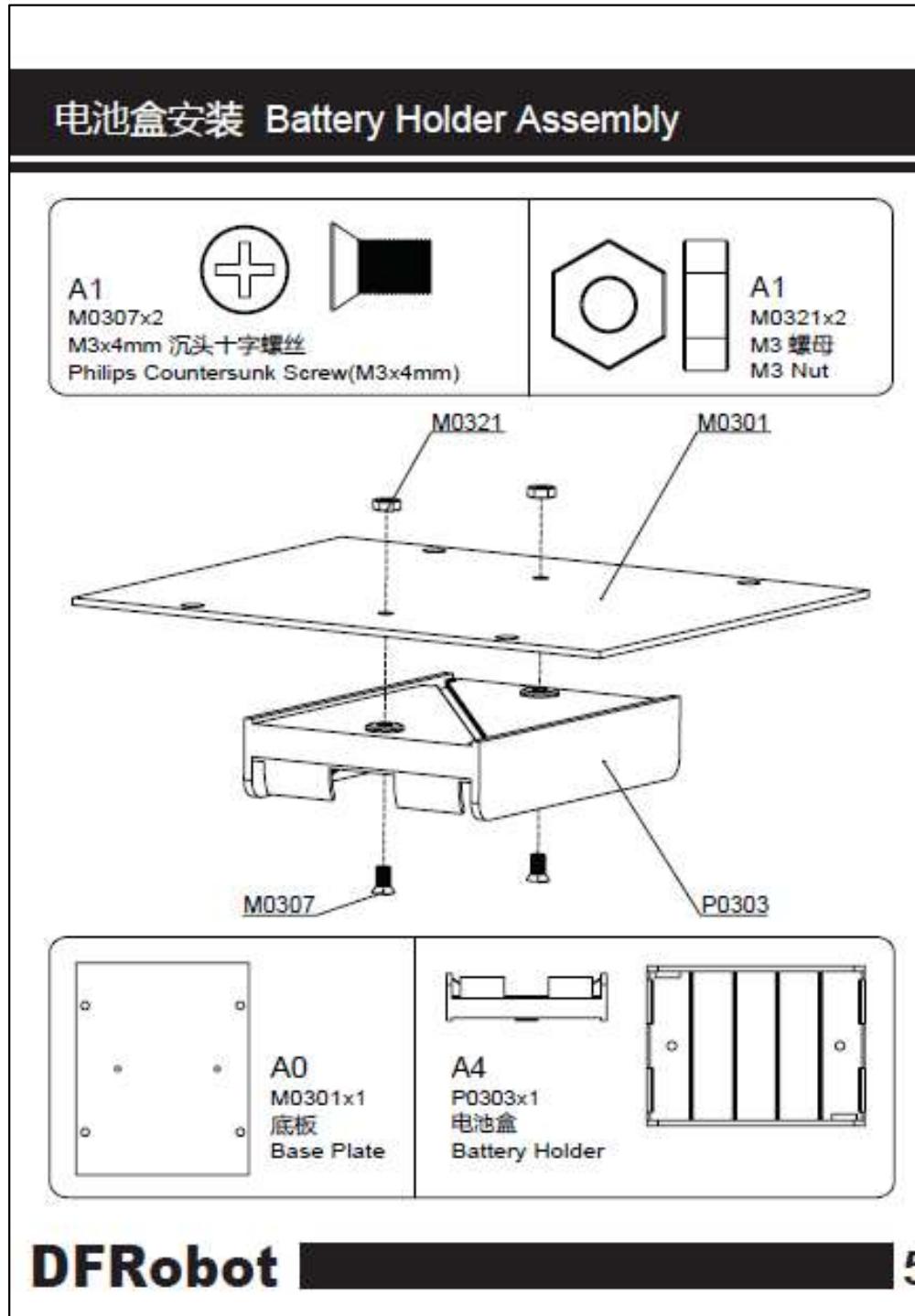
- Solder & assemble battery pack



How to solder: <https://www.youtube.com/watch?v=oqV2xU1fee8>

Assembly

- Mount AA battery pack



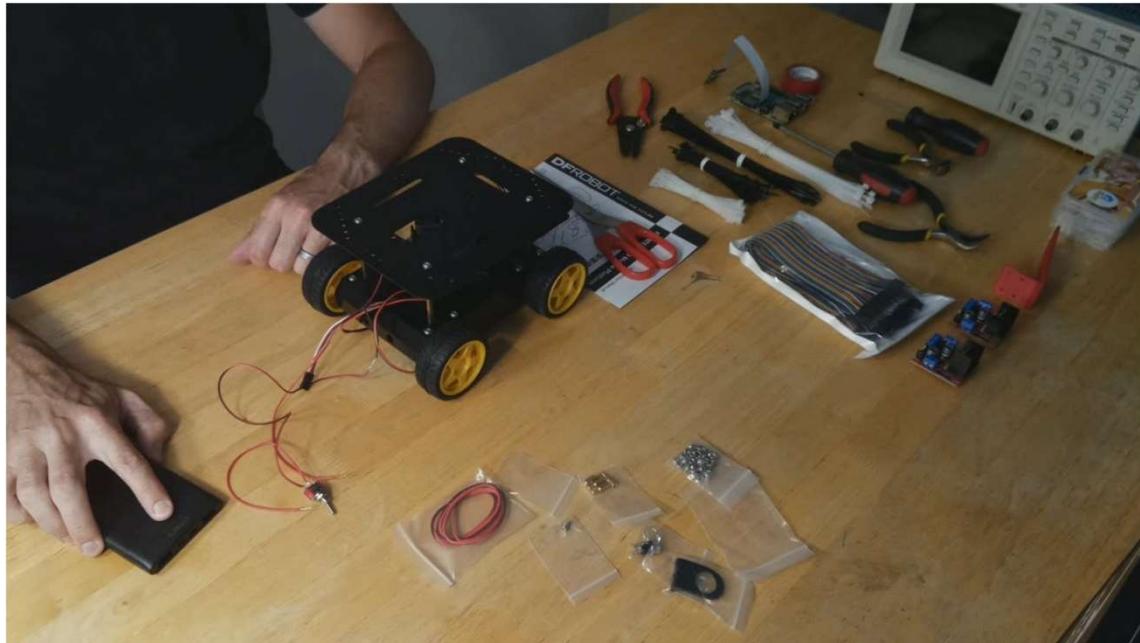
Assembly

- Mount AA battery pack



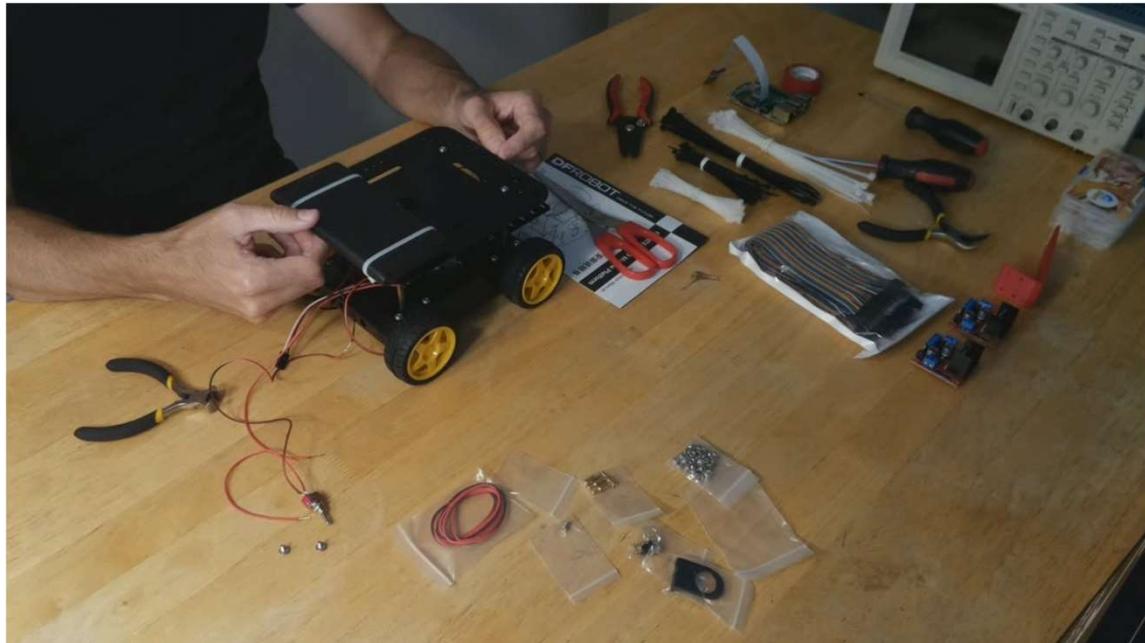
Assembly

- Mount USB battery pack



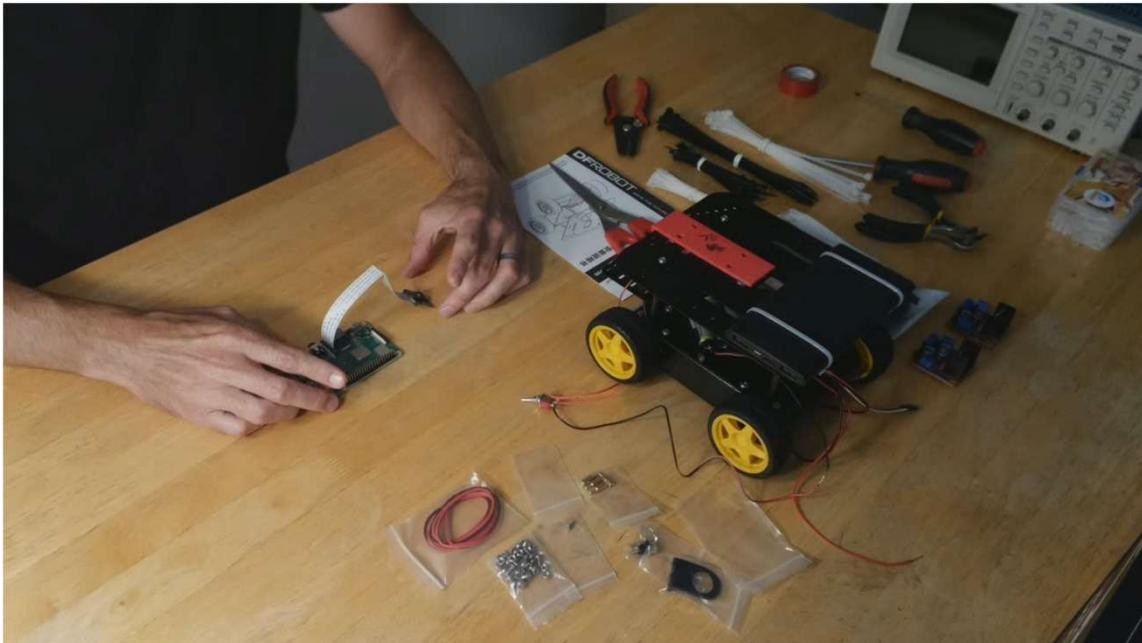
Assembly

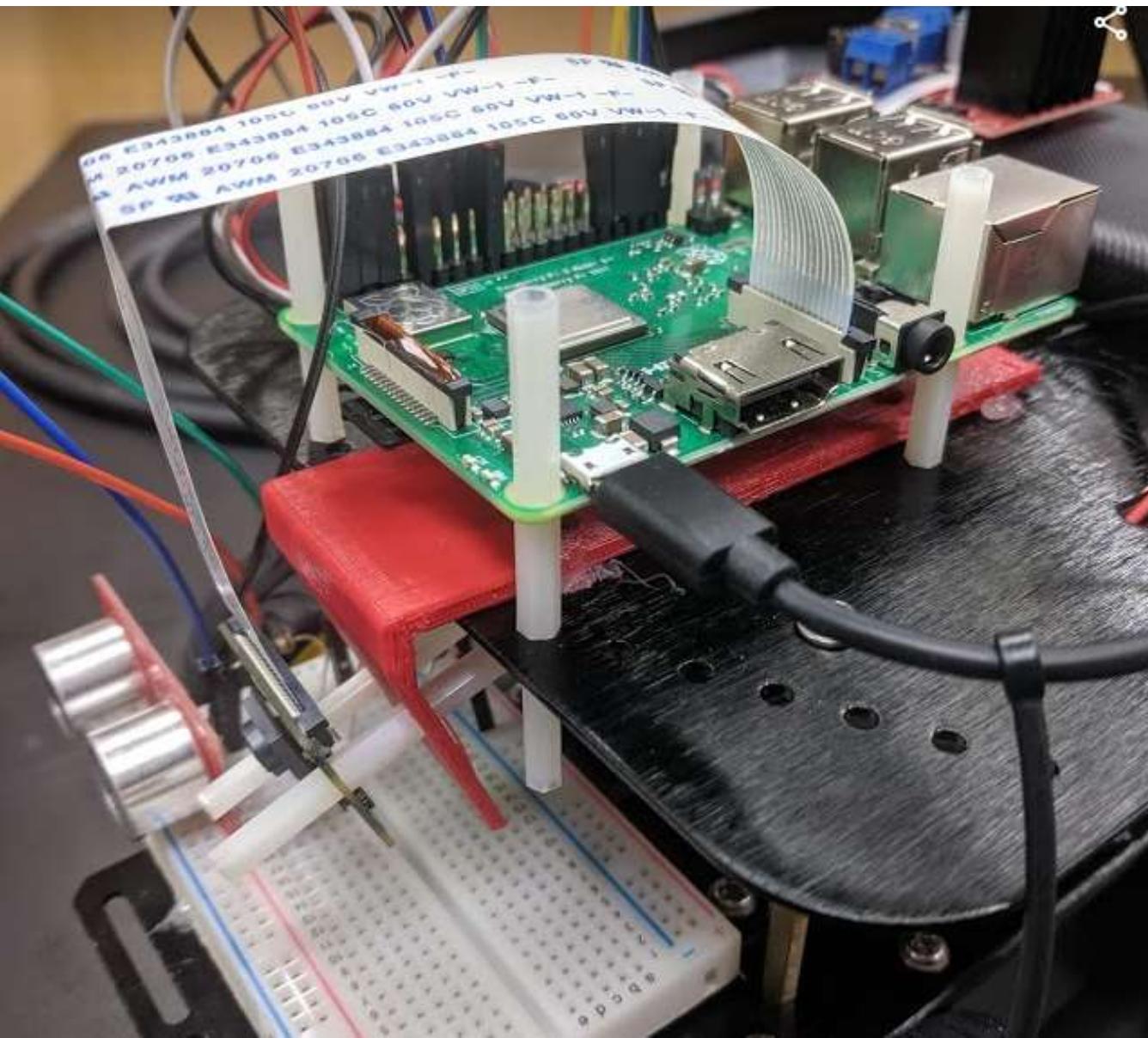
- Assemble 3D printed camera mount

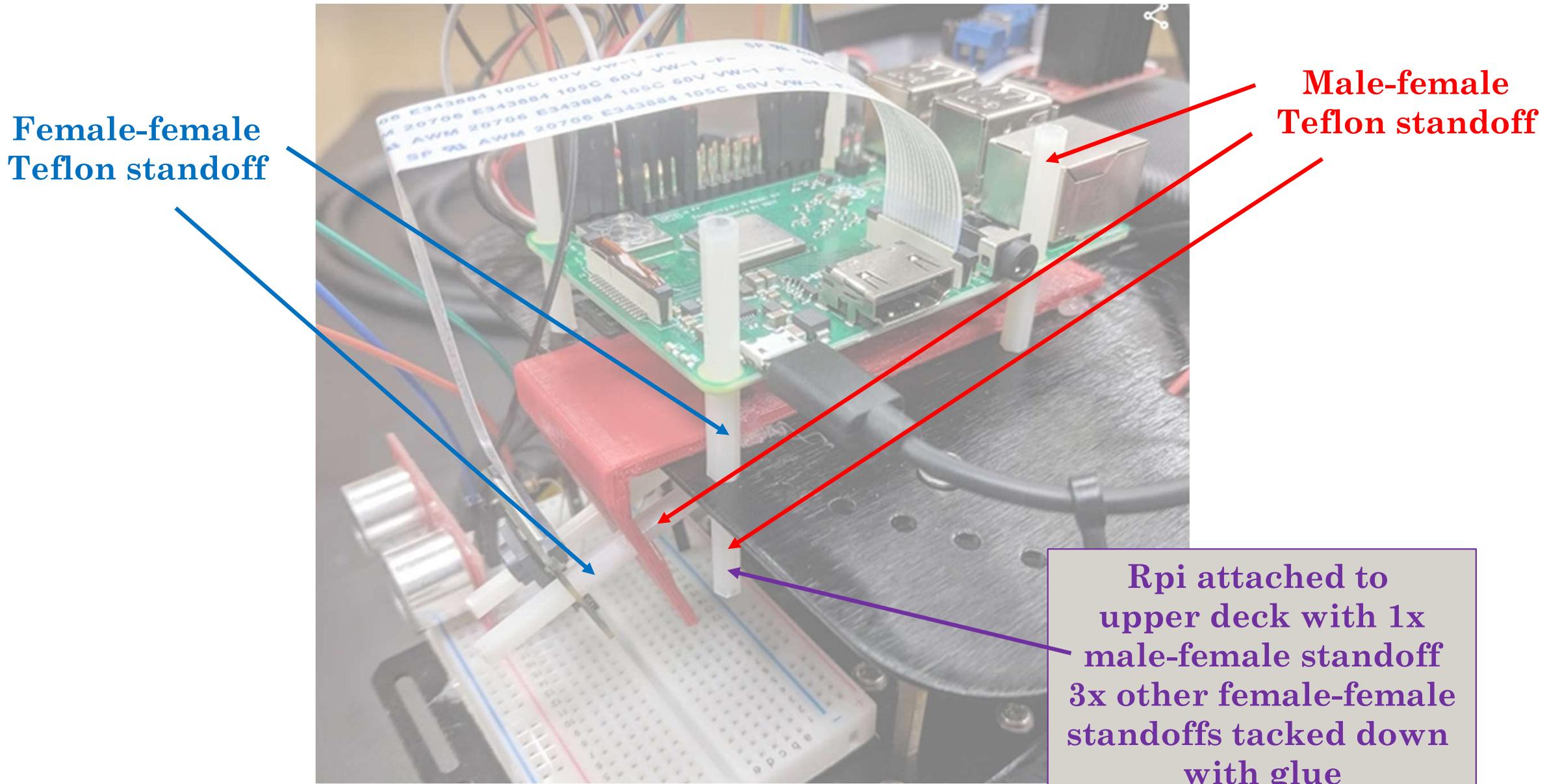


Assembly

- Mount Raspberry Pi to upper deck

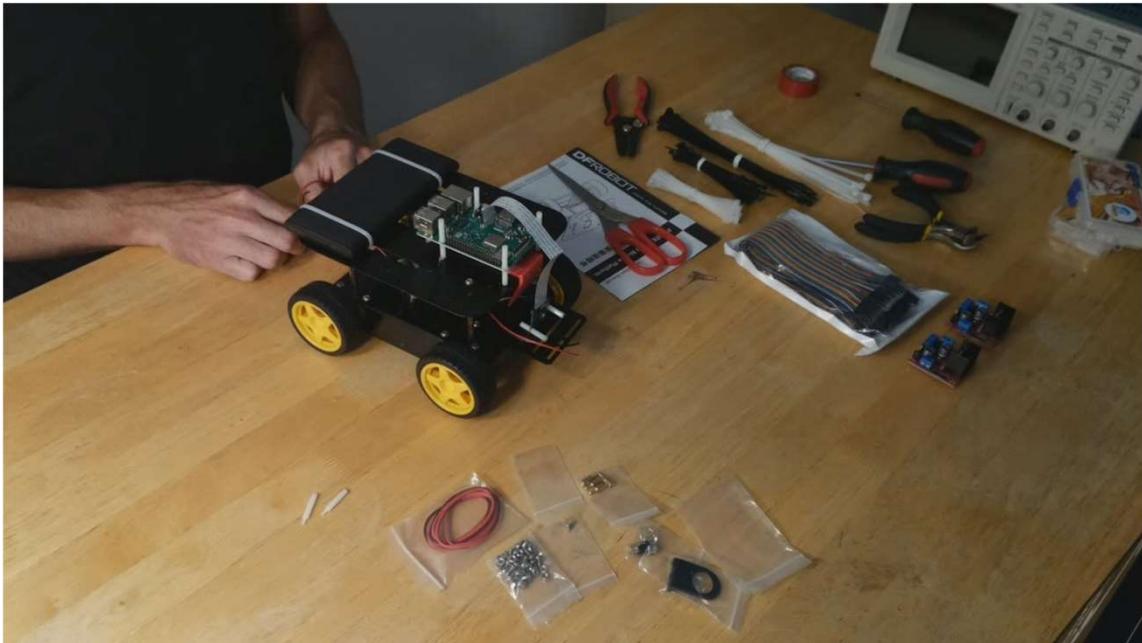






Assembly

- Mount Raspberry Pi camera to 3D printed mount

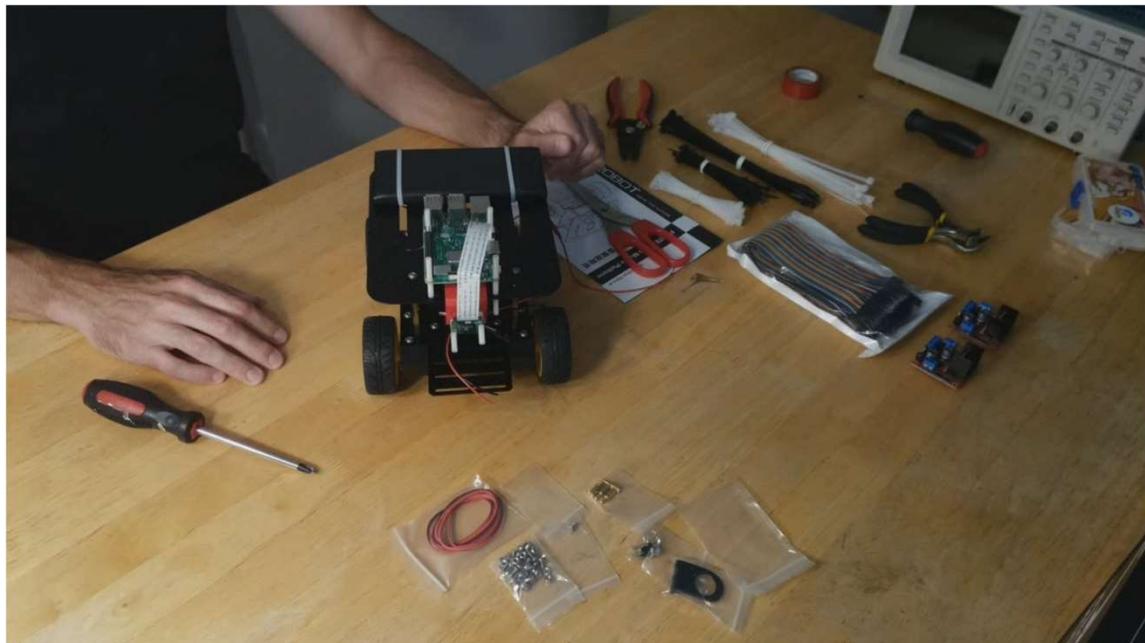


Assembly

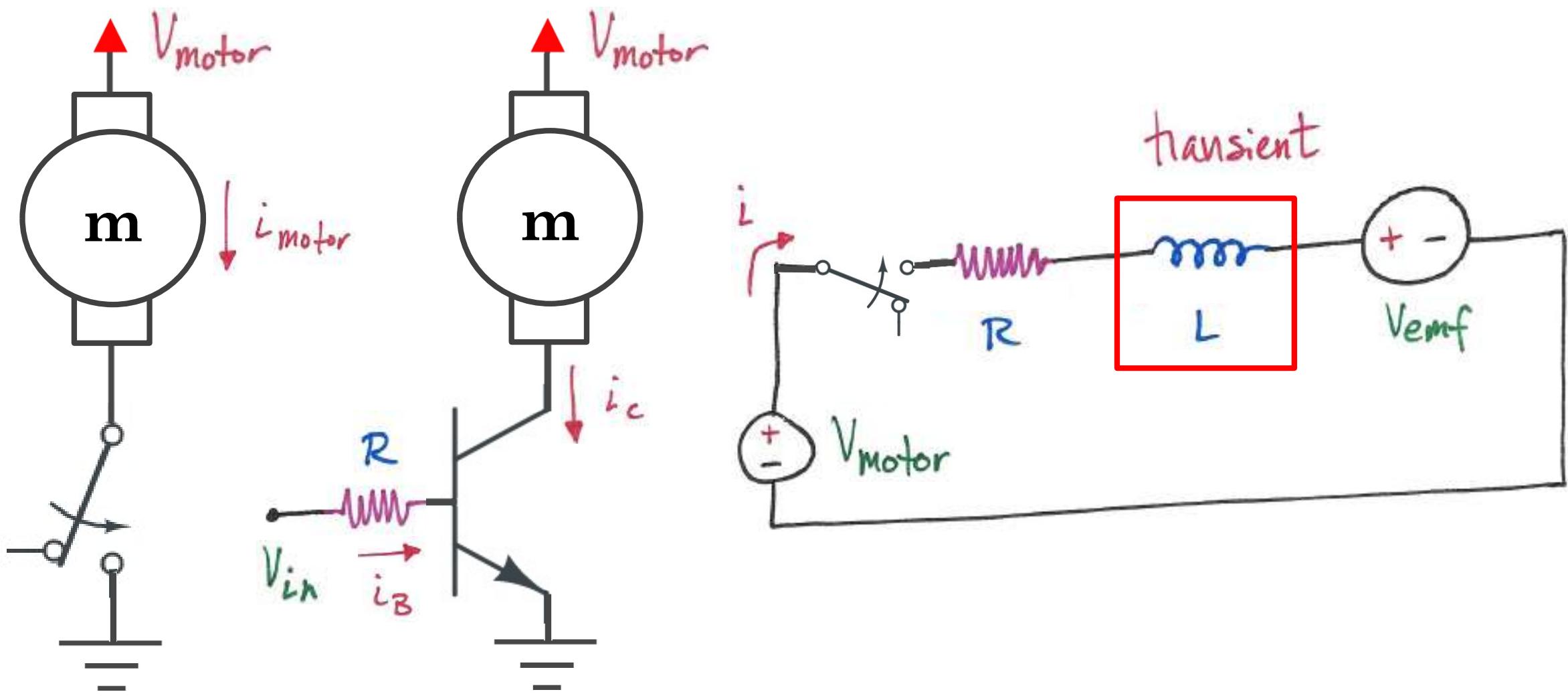
- Mount H-bridges & zip tie power switch to upper deck

***Simply fasten the zip tie(s) until
the H-bridge is snug**

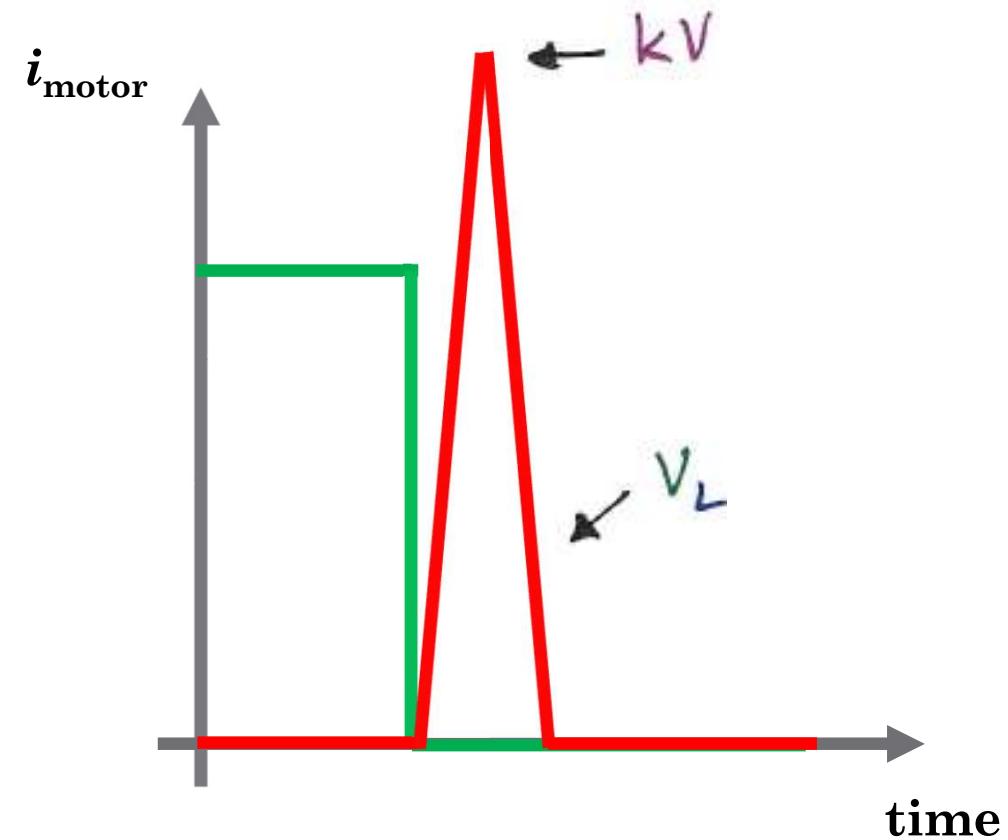
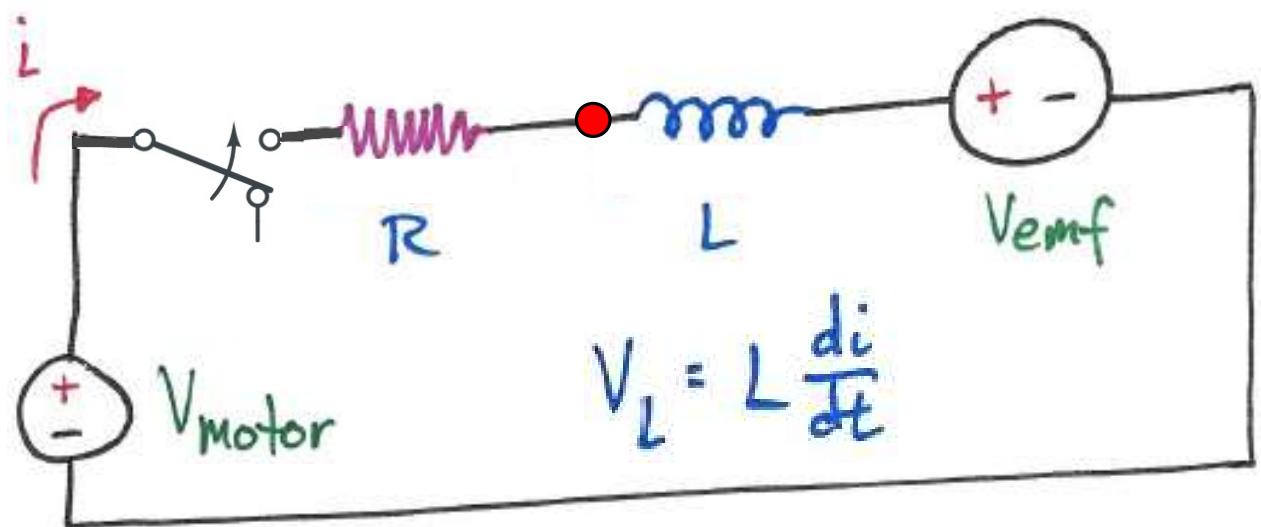
...tightening too far can bend the pins
and/or the board



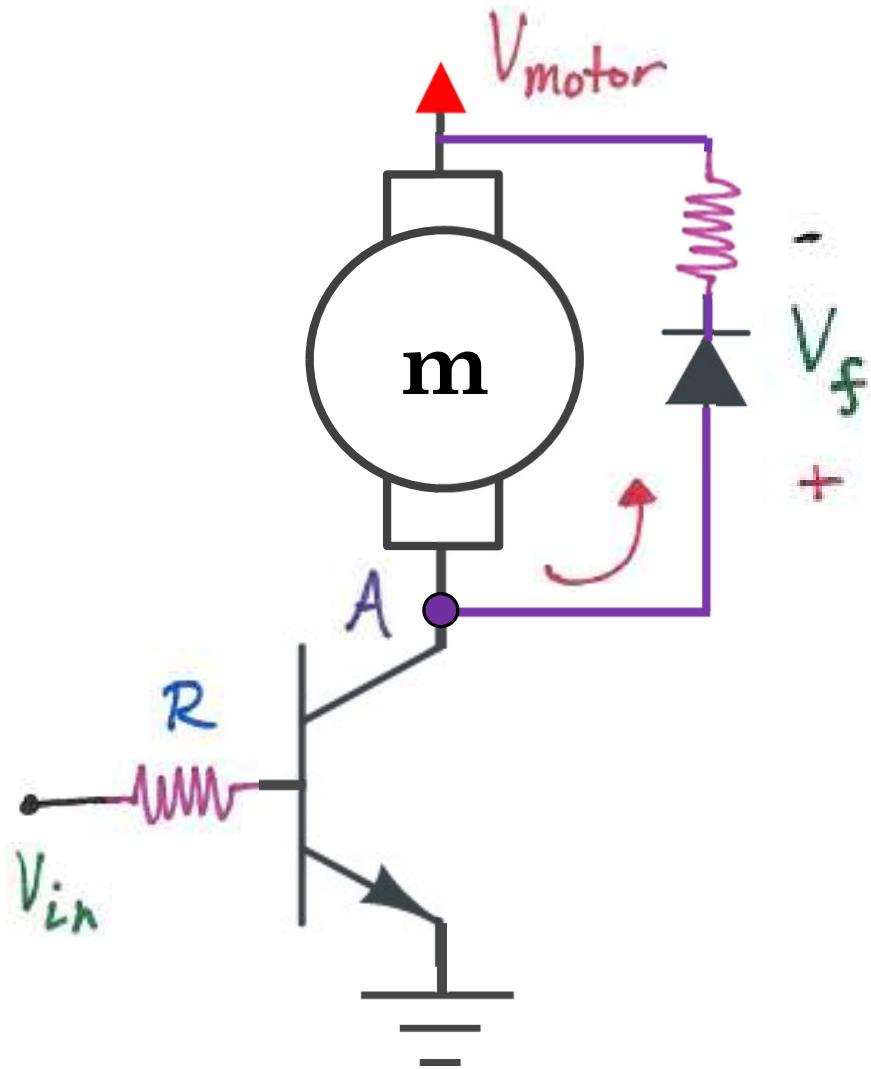
Locomotion



What about the inductor in the motor?

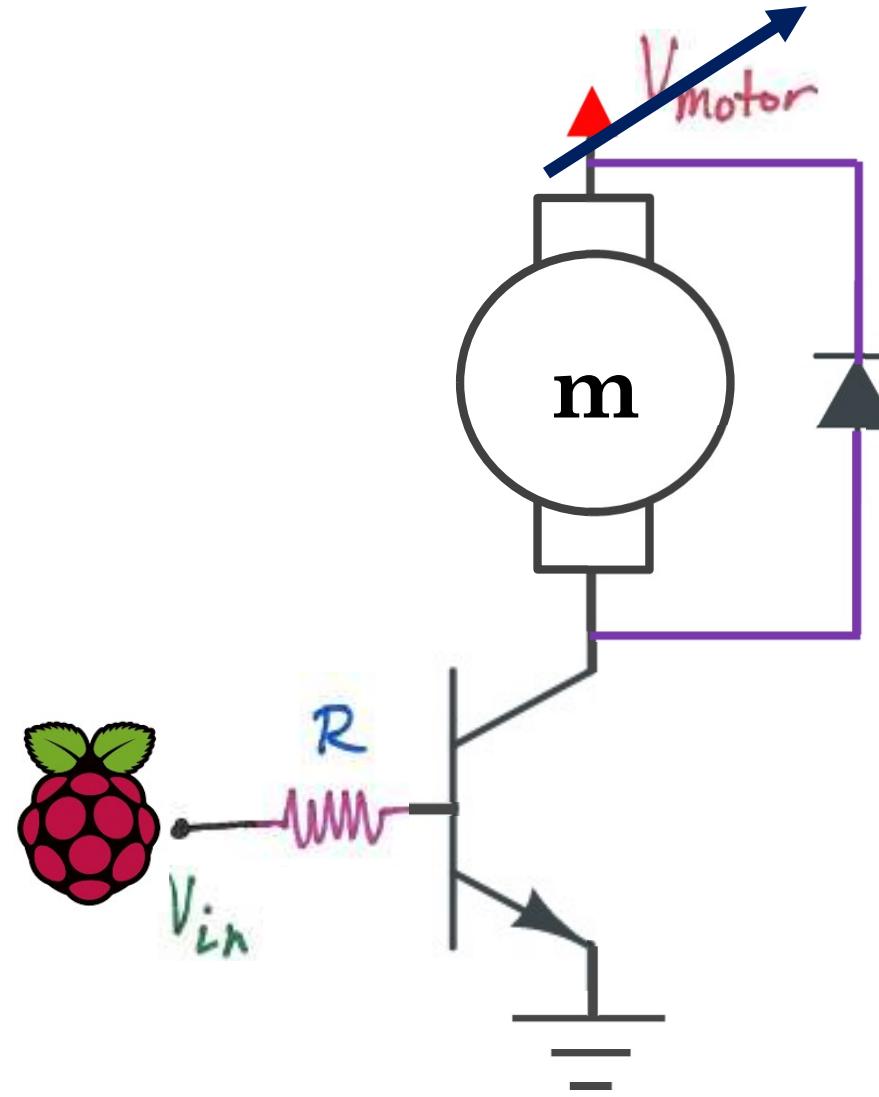
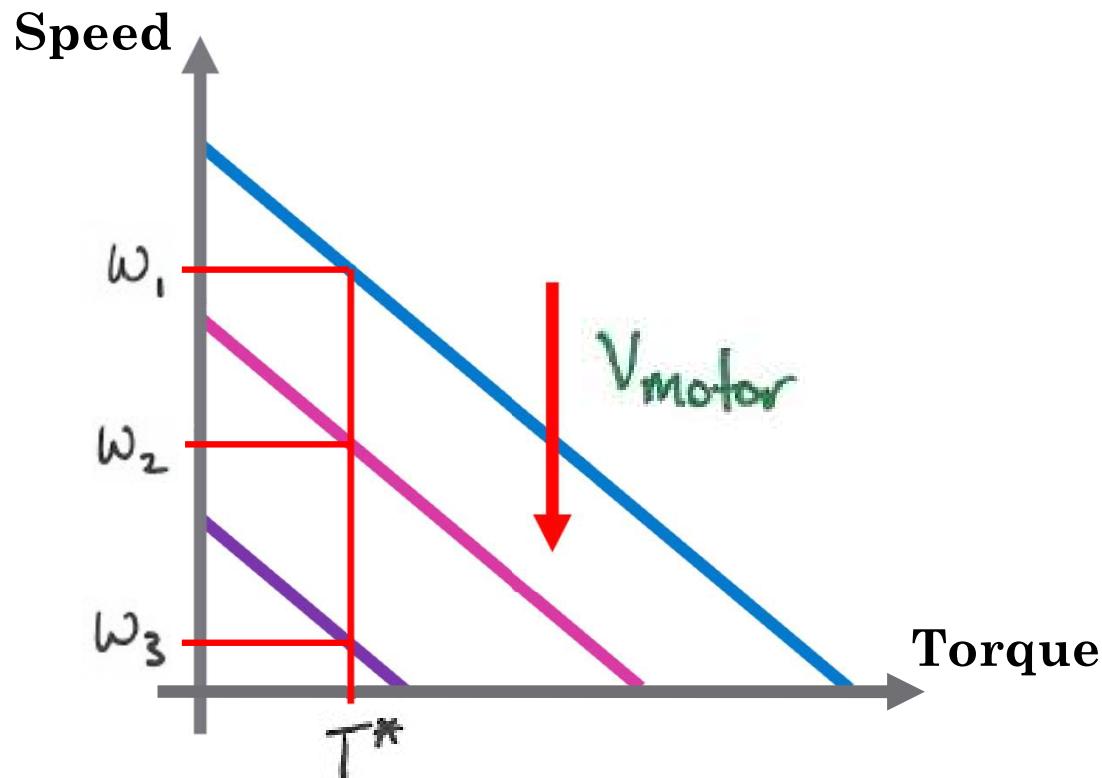


Flyback diode

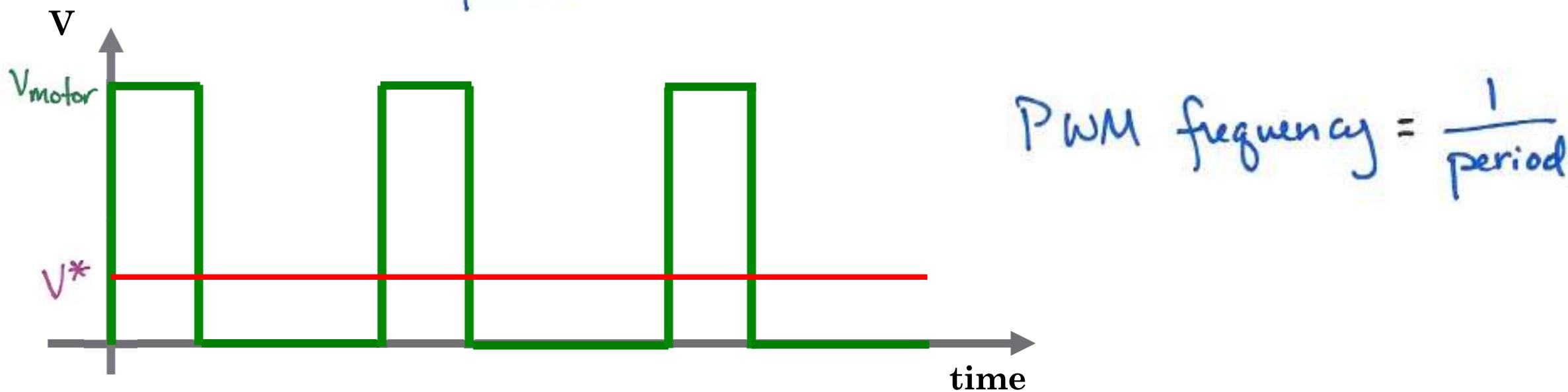
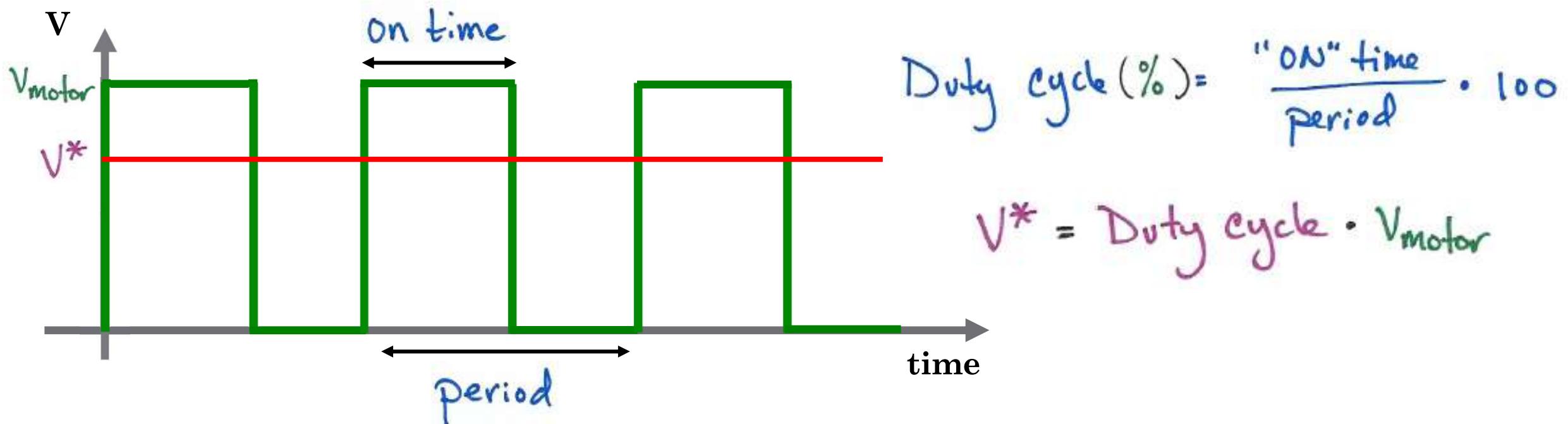


- Diode is a **one-way** current valve
 - no affect in normal operation
 - diode allows current to flow when voltage @ A is $V_{motor} + V_f$

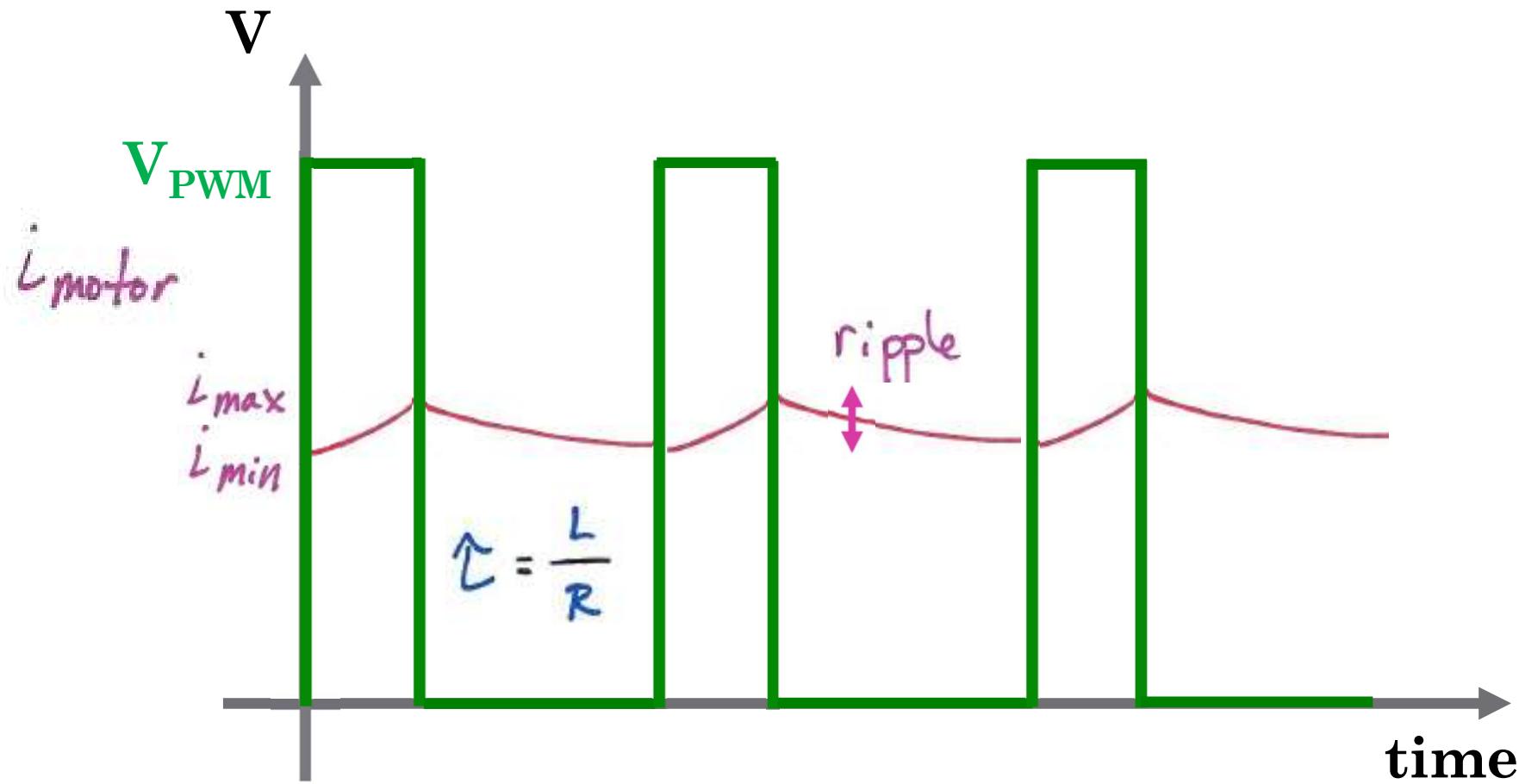
Controlling speed via variable voltage

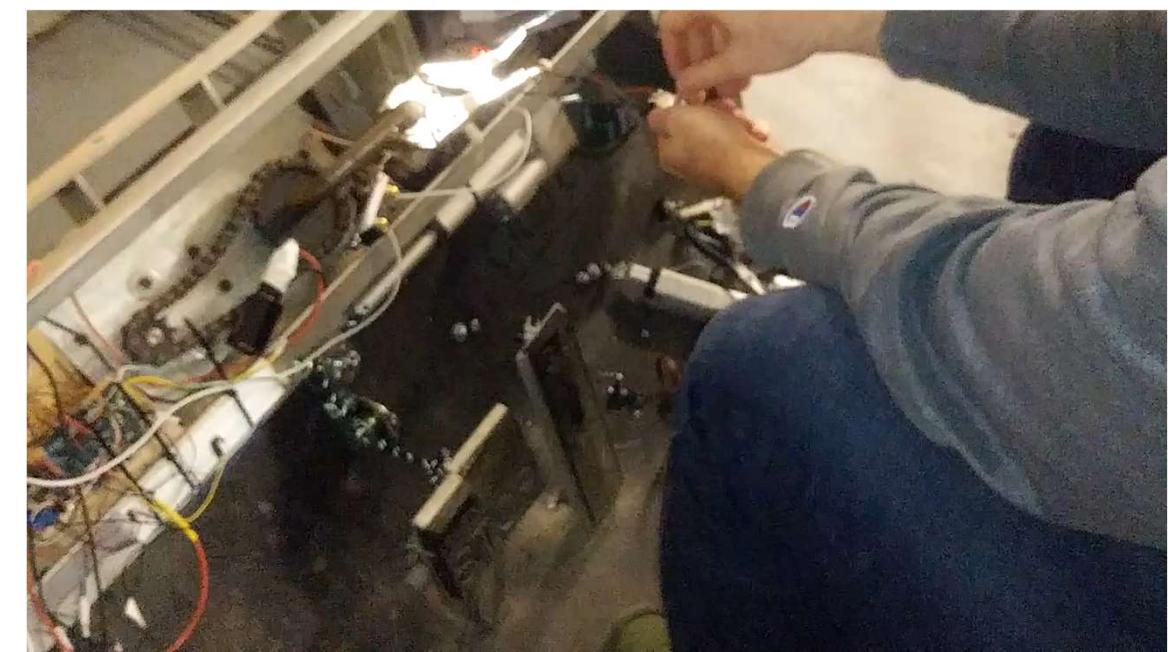
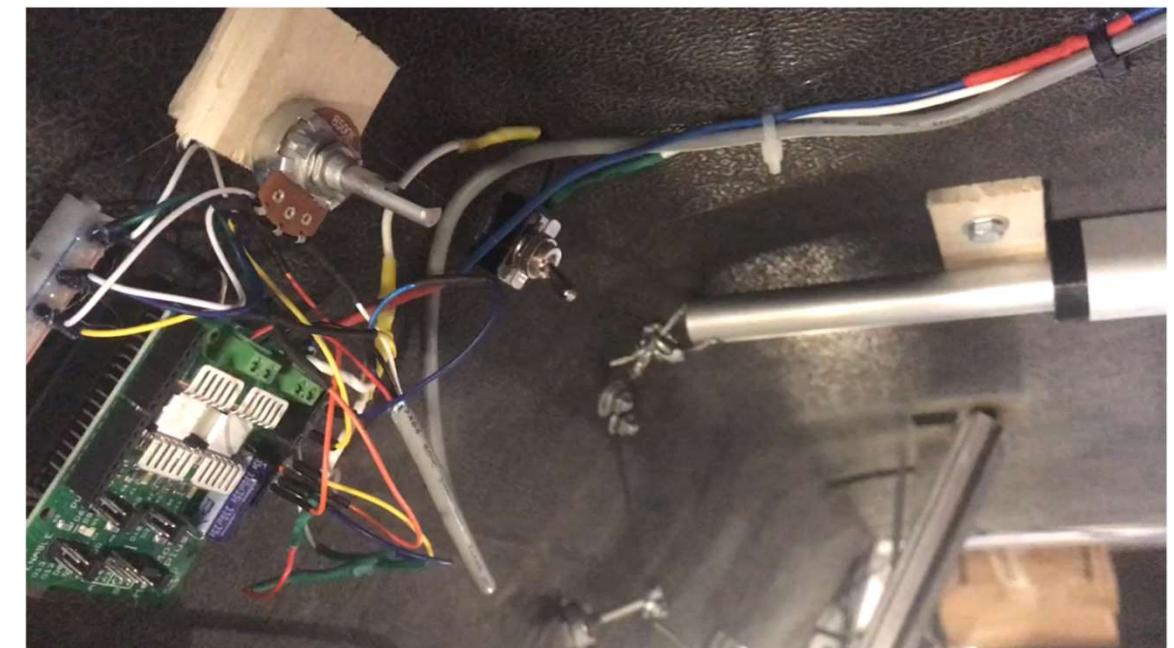


Variable voltage via PWM



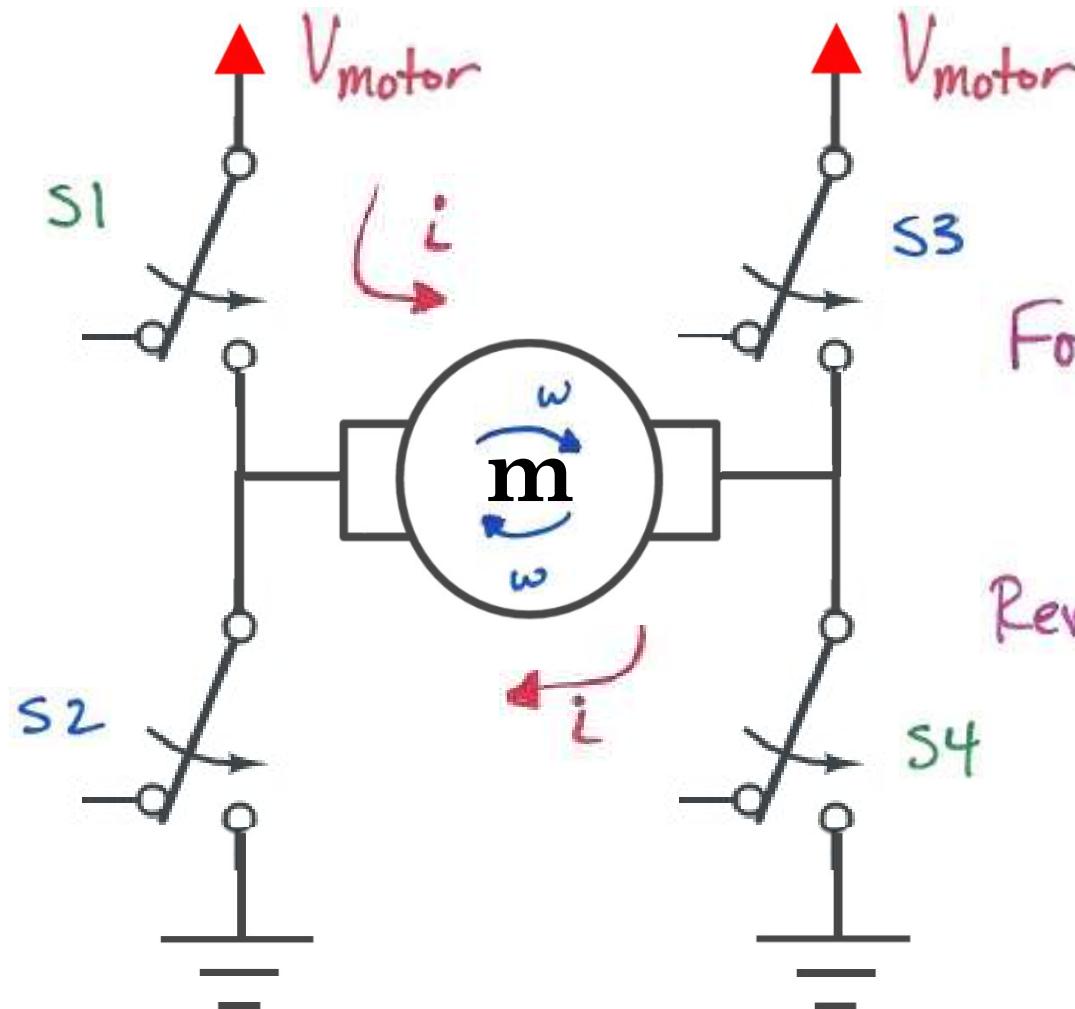
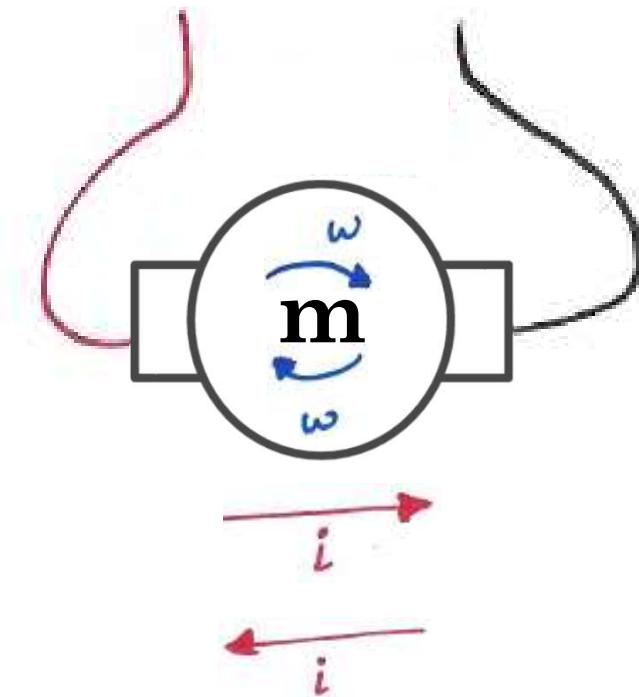
Torque ripple





Basic direction control

"H-Bridge"



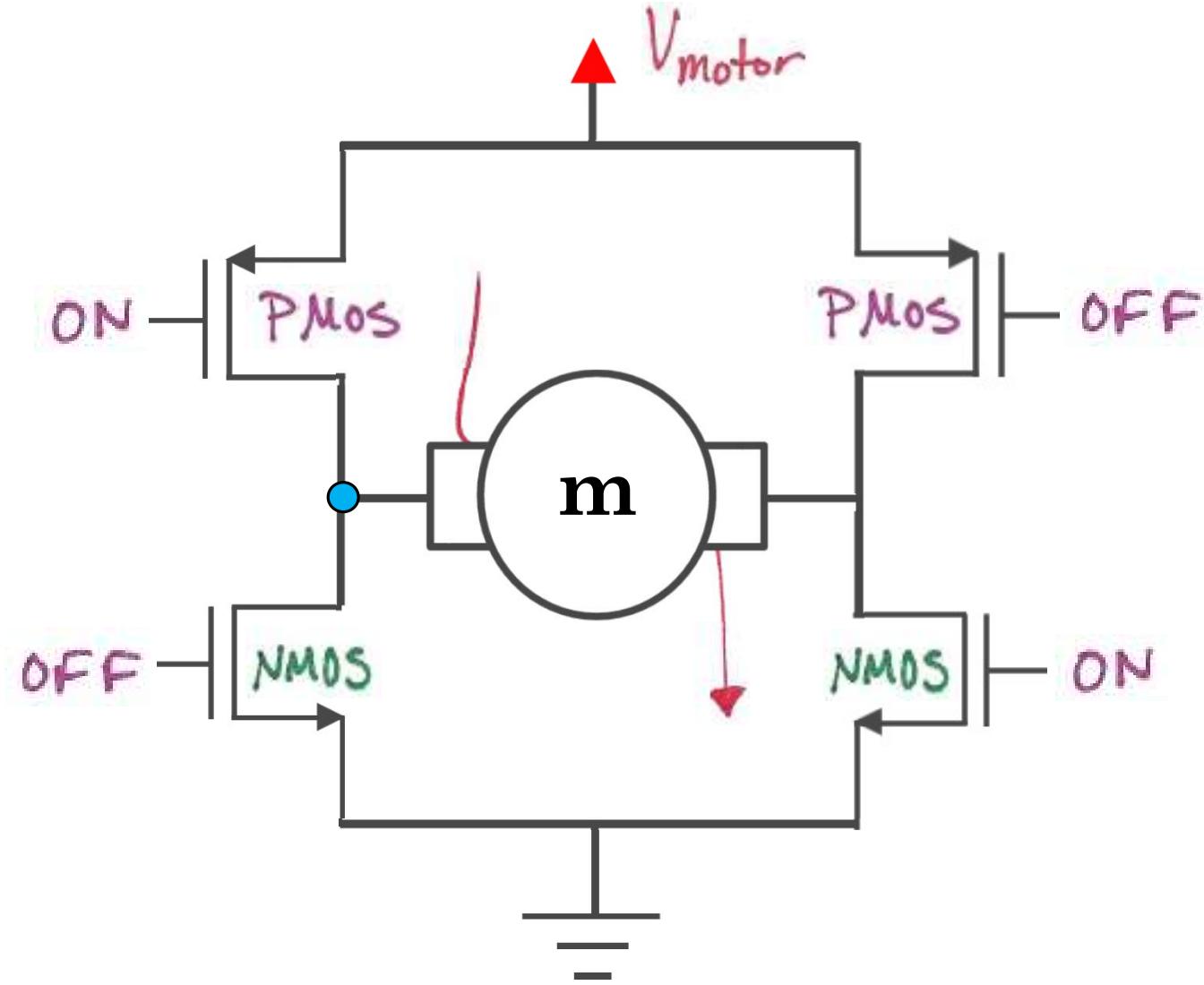
Forward : S1/S4 closed

S2/S3 open

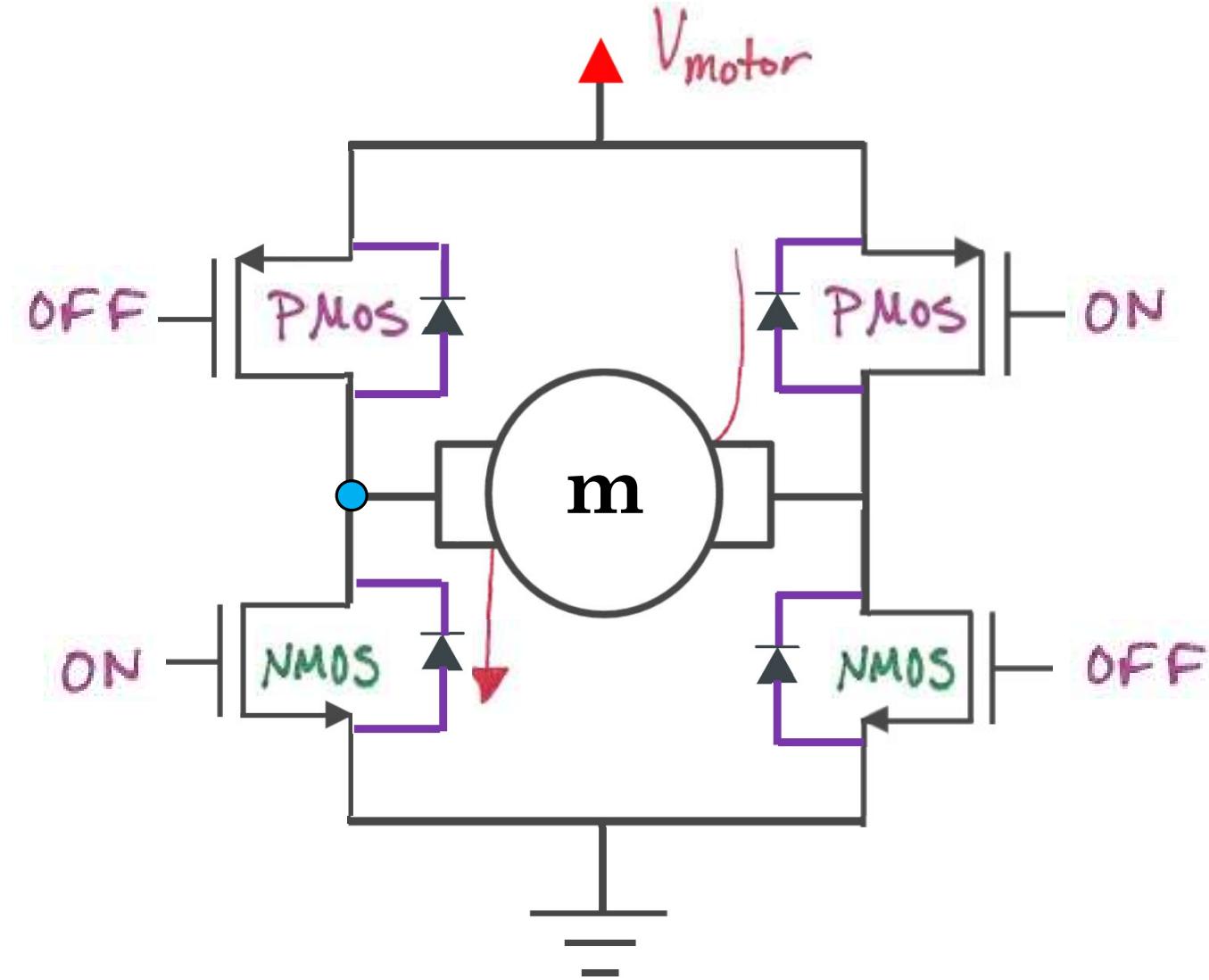
Reverse : S2/S3 closed

S1/S4 open

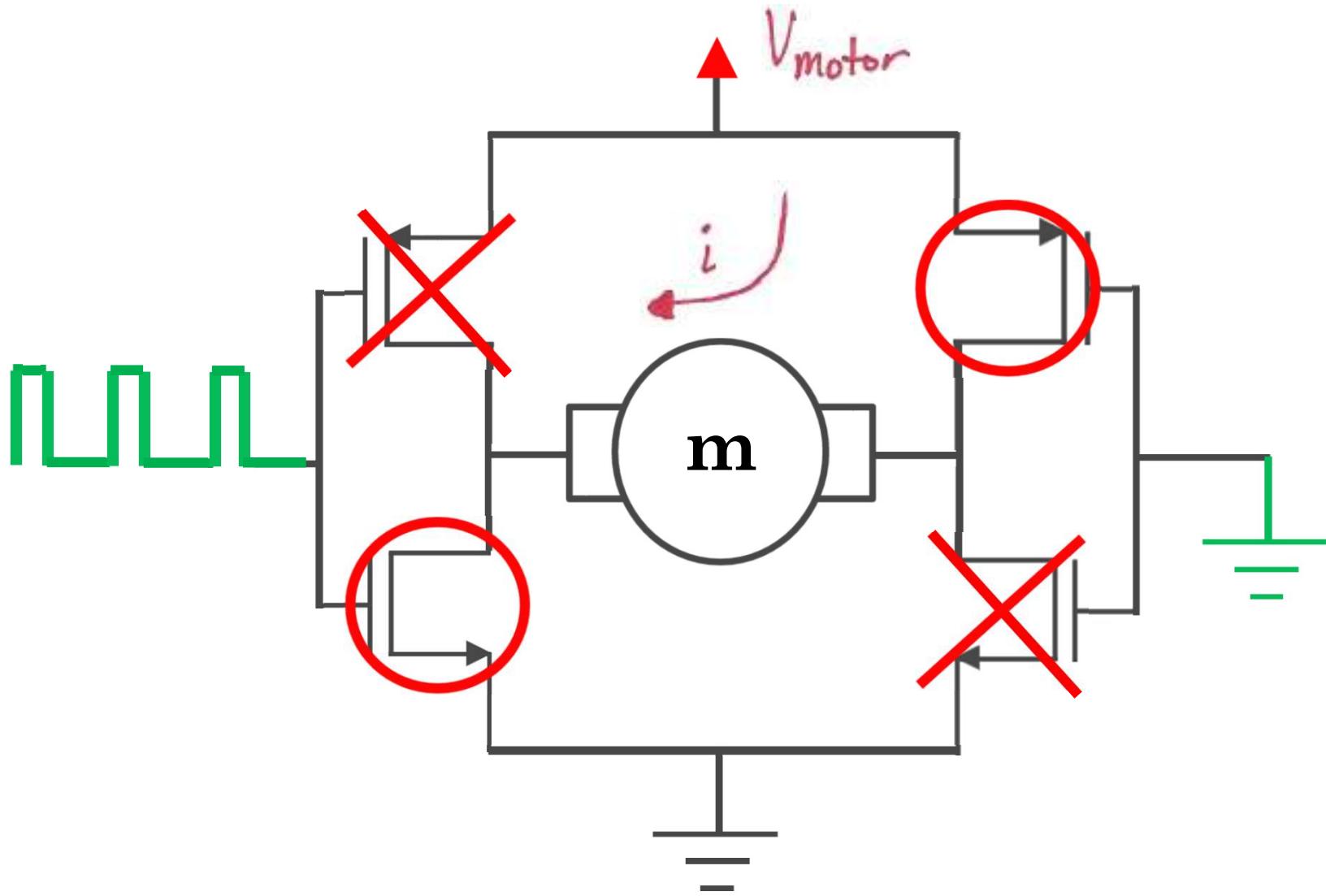
Transistor H-Bridge



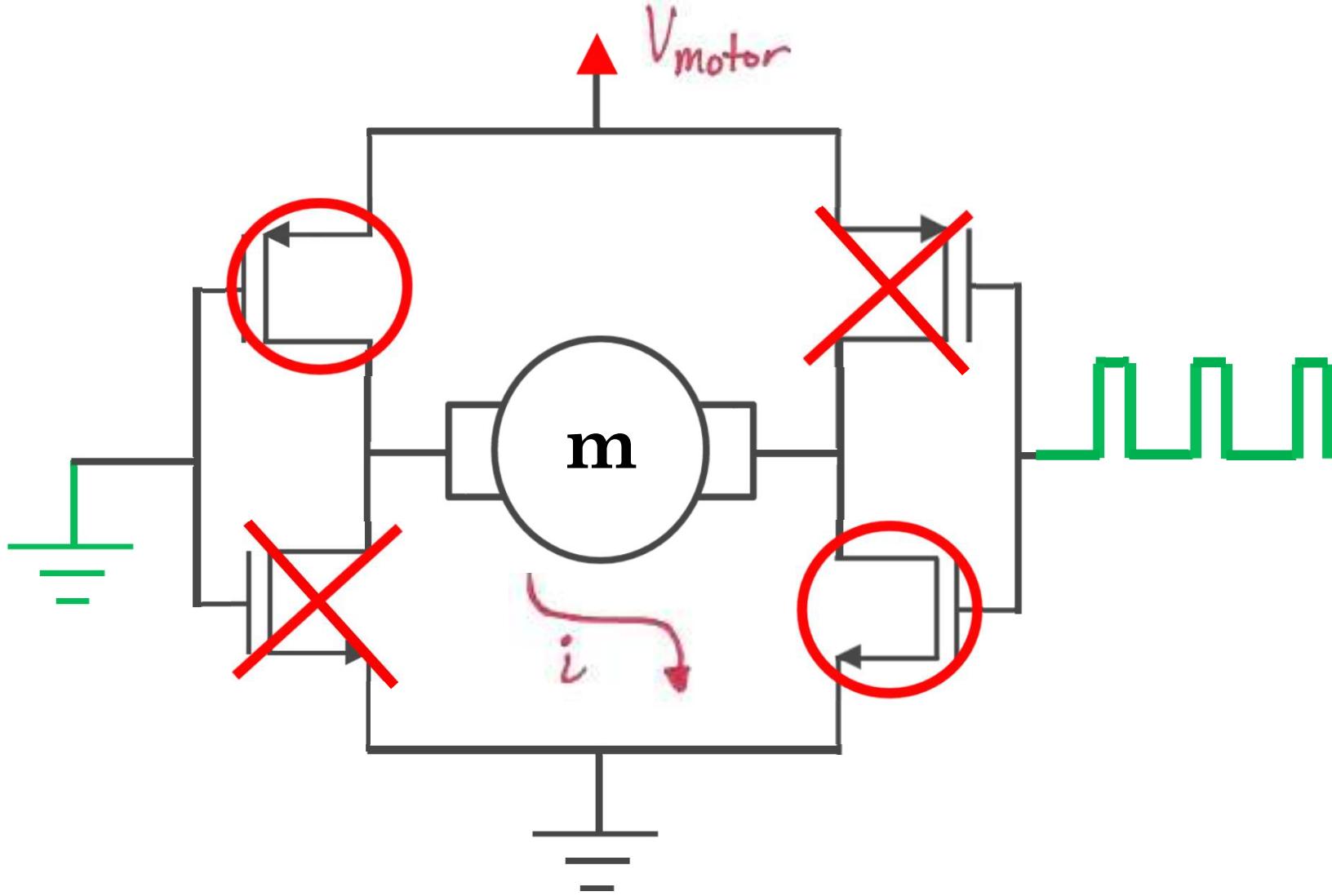
Transistor H-Bridge



Transistor H-Bridge

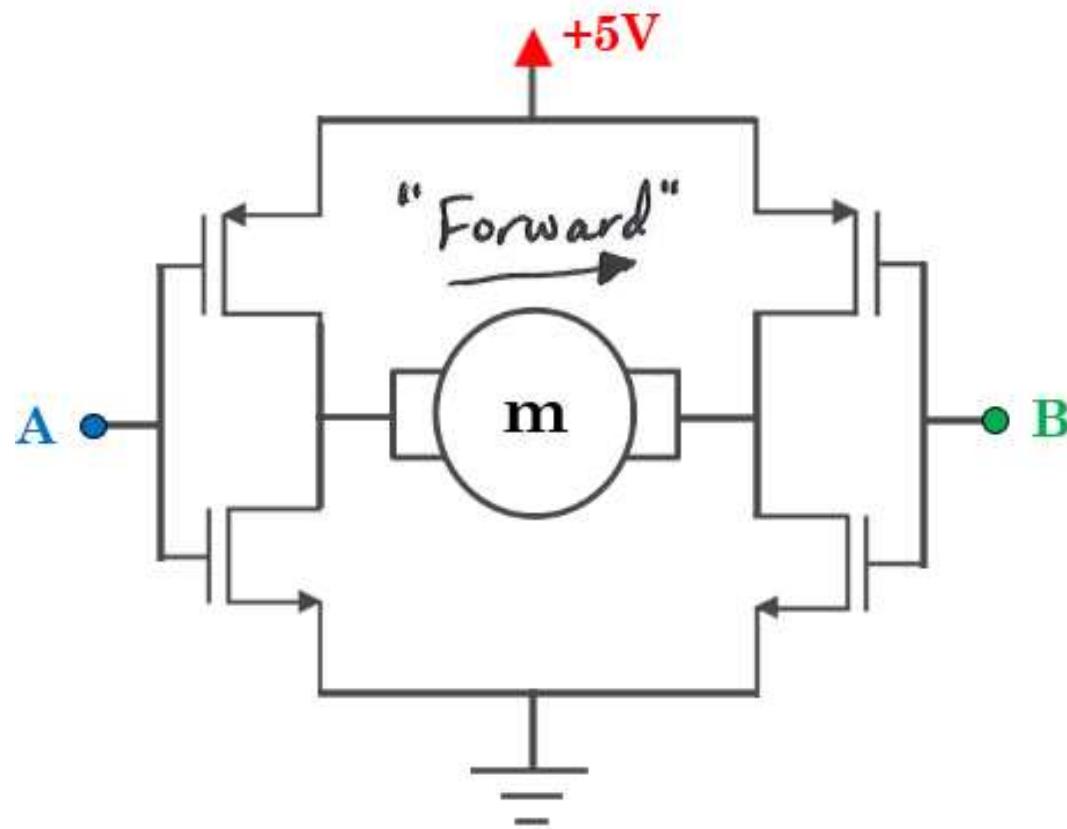


Transistor H-Bridge



In-Class Exercise

- Using the circuit below and a Raspberry Pi to drive the motor, **sketch the time-domain signals** required at **A** and **B** to turn forward at 30% full speed and reverse at 70% full speed. Assume speed is proportional to voltage, the threshold voltage for all transistors is 3V, and a V_{PWM} period of 1 msec.

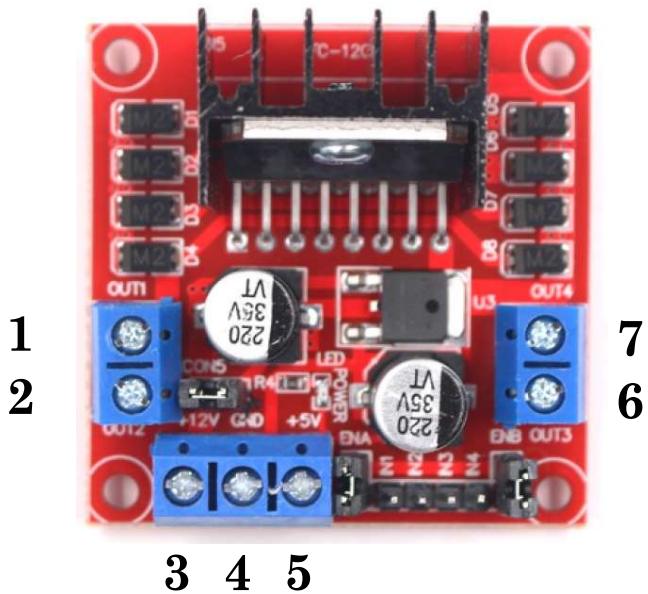


In-Class Exercise

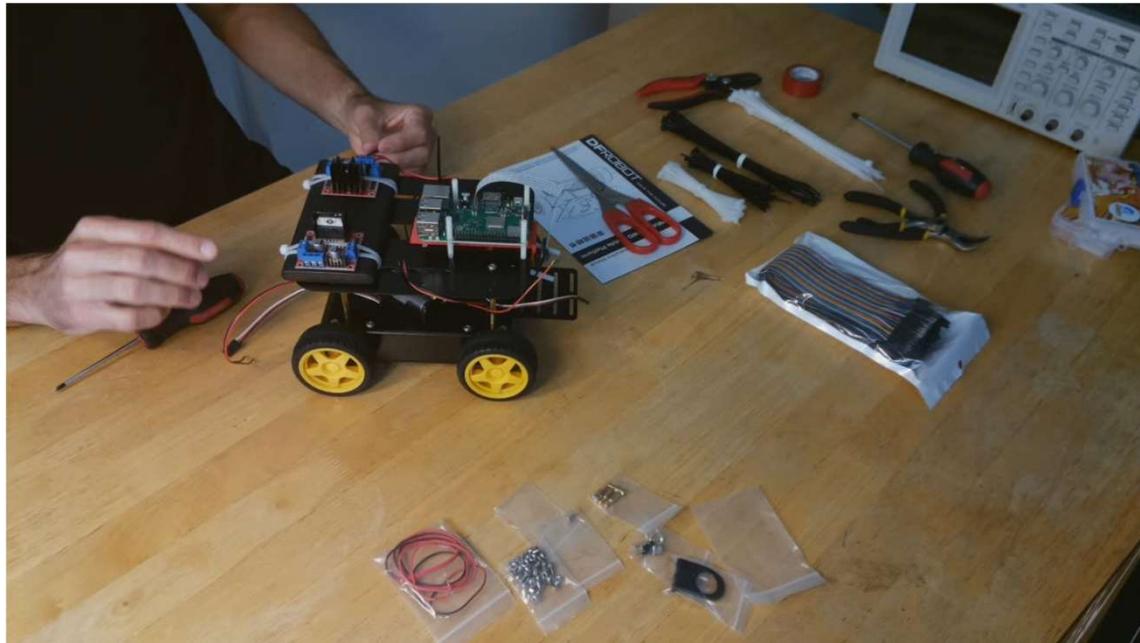


Assembly

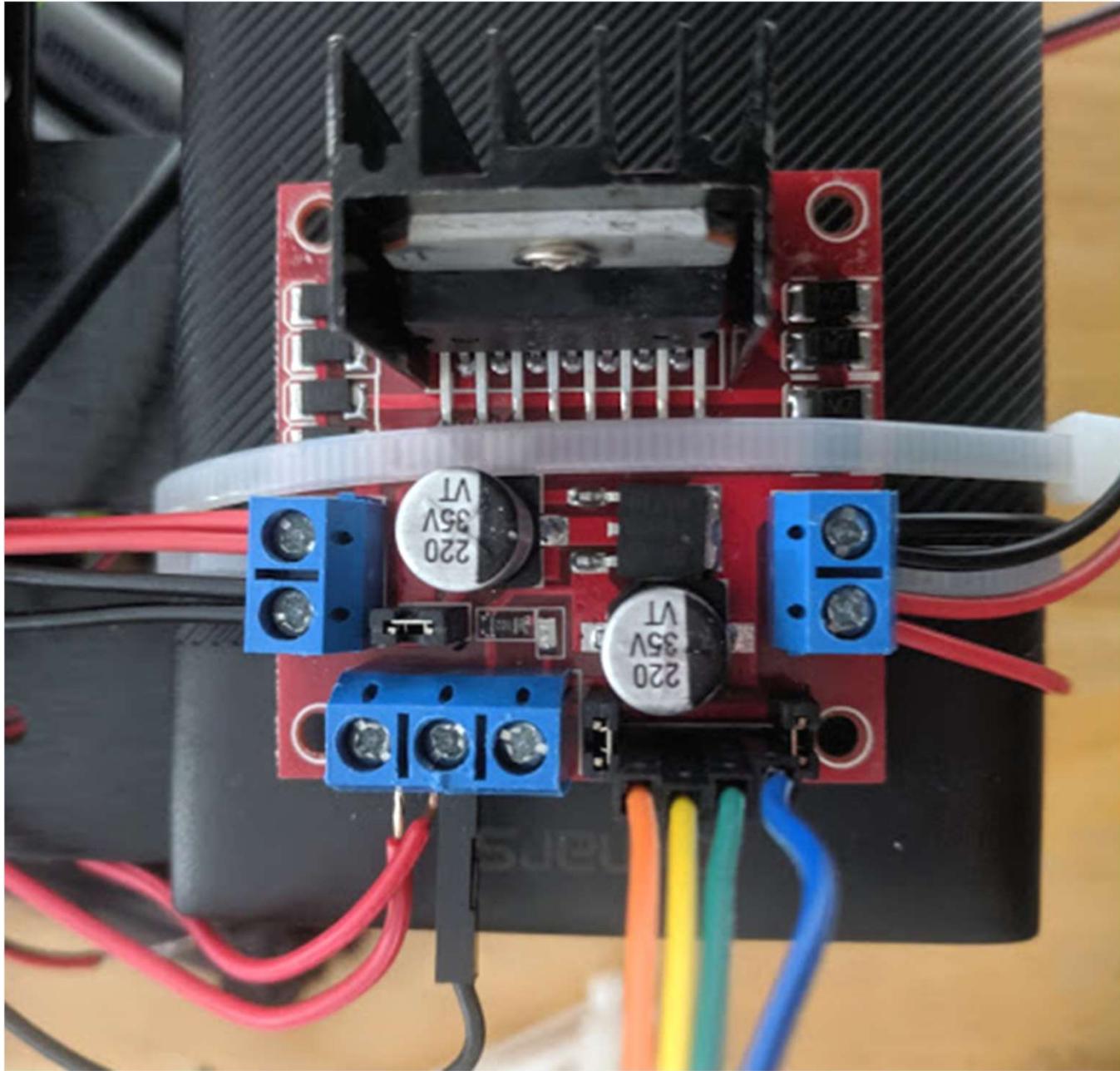
- Wire motors to H-bridge



- 1: "+" voltage, left motors
- 2: "-" voltage, left motors
- 3: **power in** from AA battery pack
- 4: GND

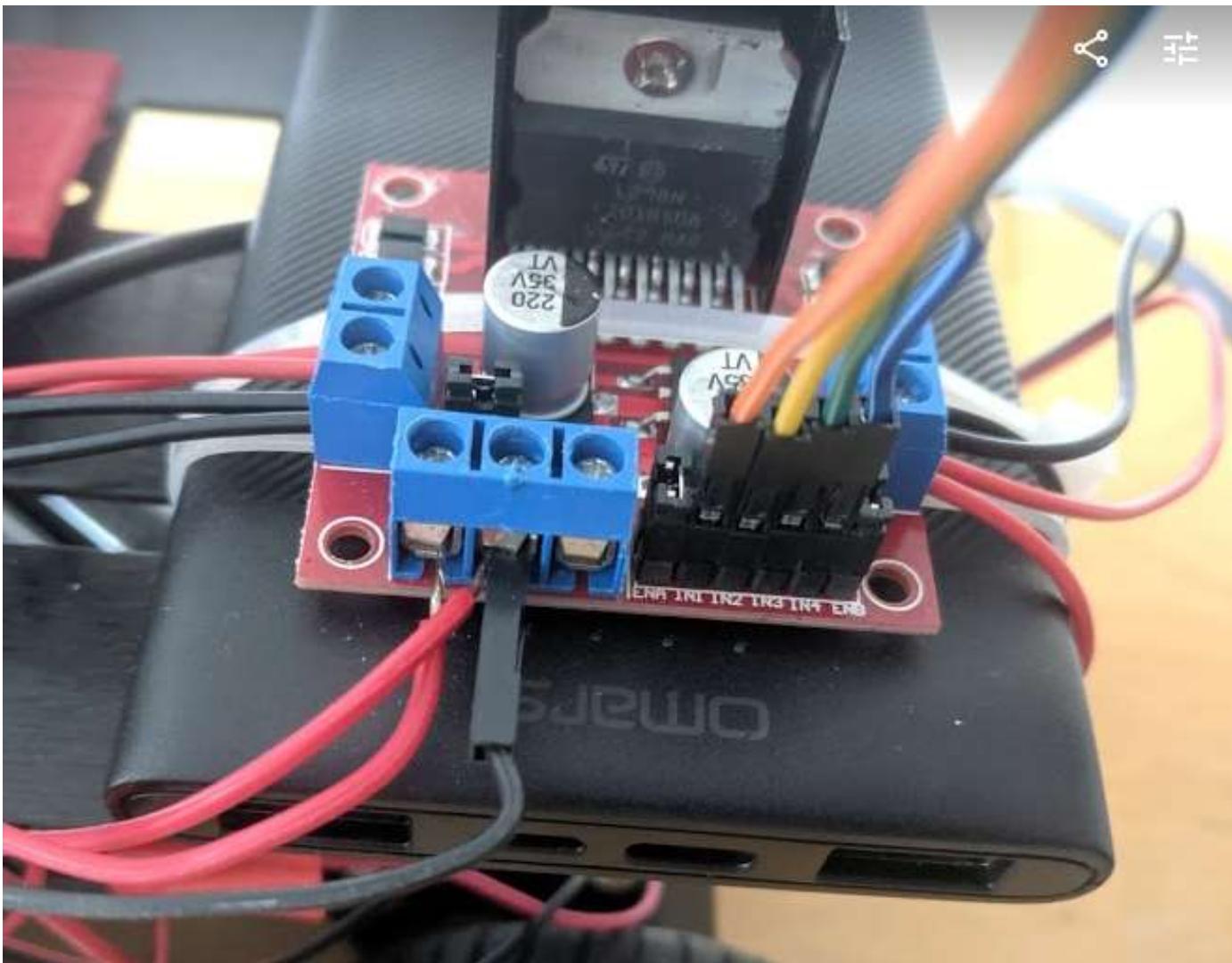


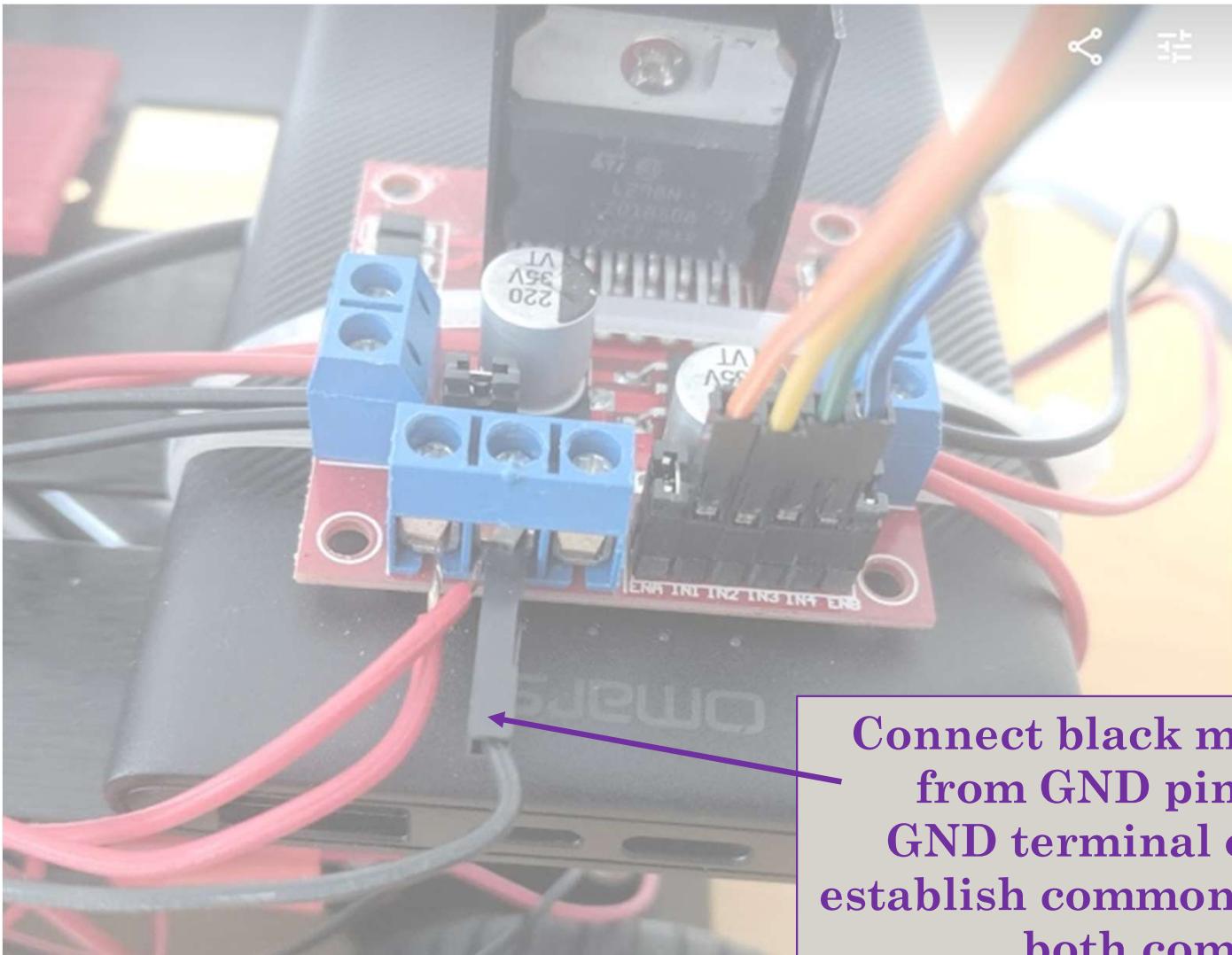
- 5: 5V output
- 6: "+" voltage, right motors
- 7: "-" voltage, right motors



- 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
- 6: "+" voltage, right motors
- 7: "-" voltage, right motors

3 4 5

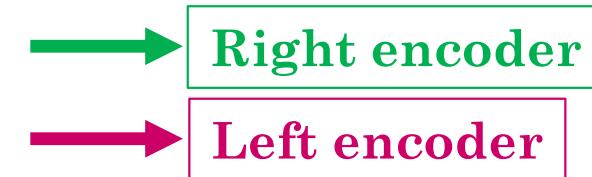
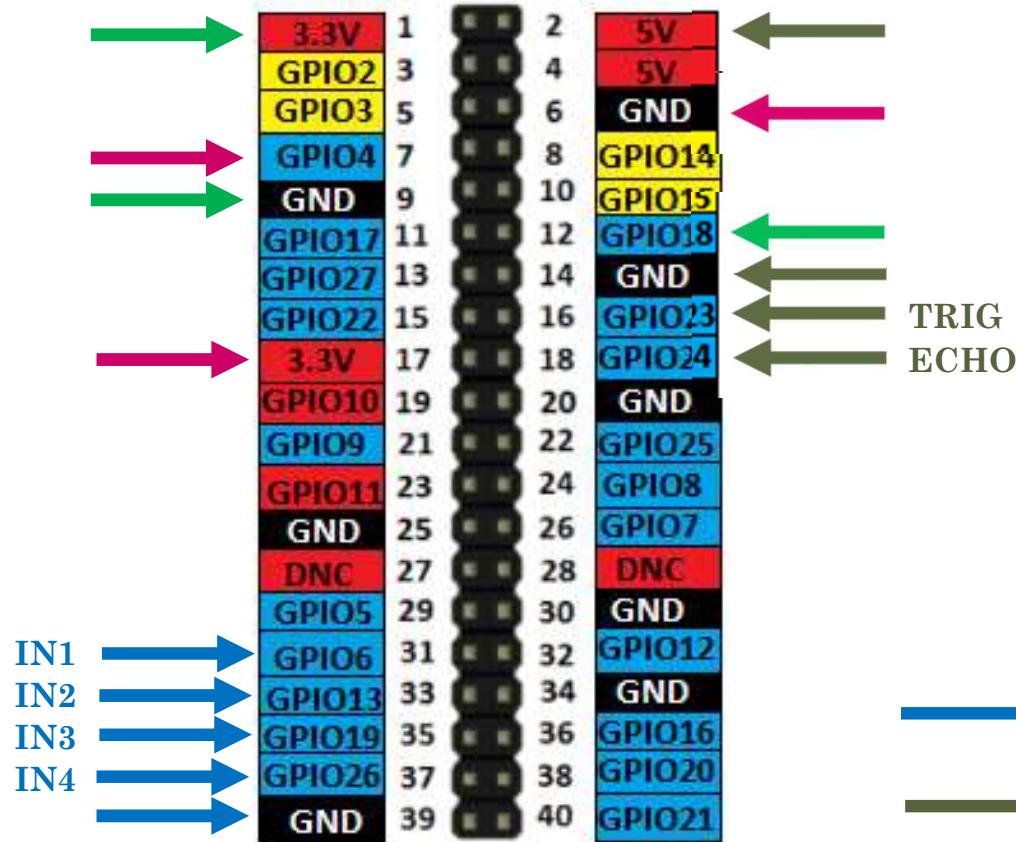




Connect black male-female wire
from GND pin 39 on Rpi to
GND terminal on H-bridge to
establish common ground between
both components

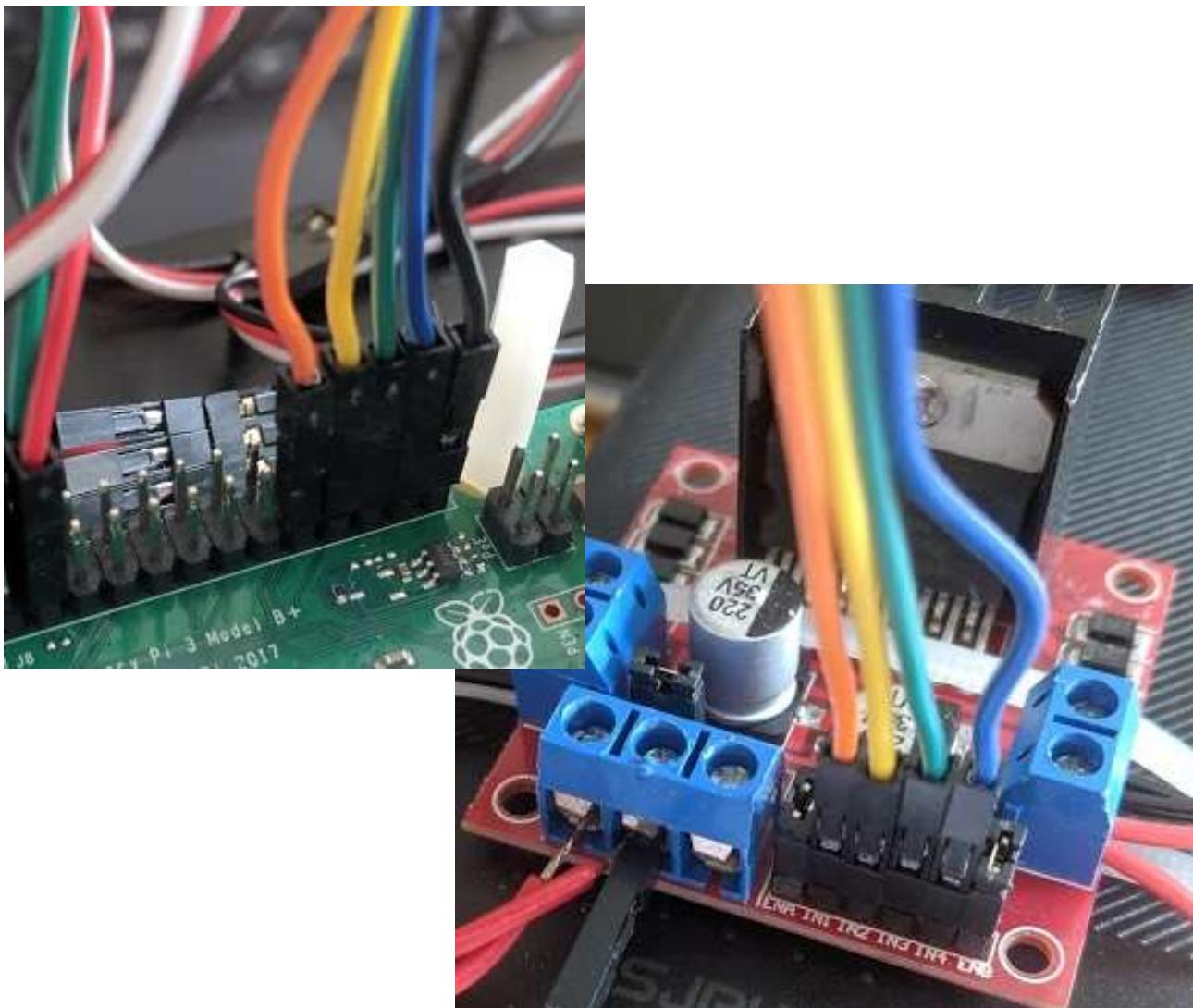
Assembly

- Wire H-bridge to Raspberry Pi



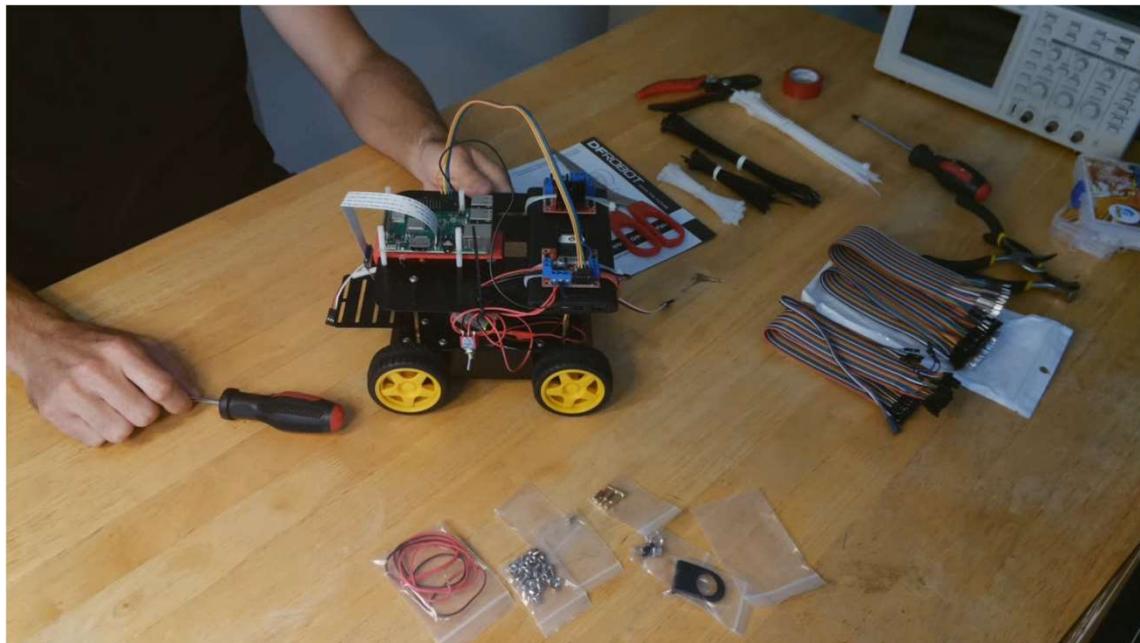
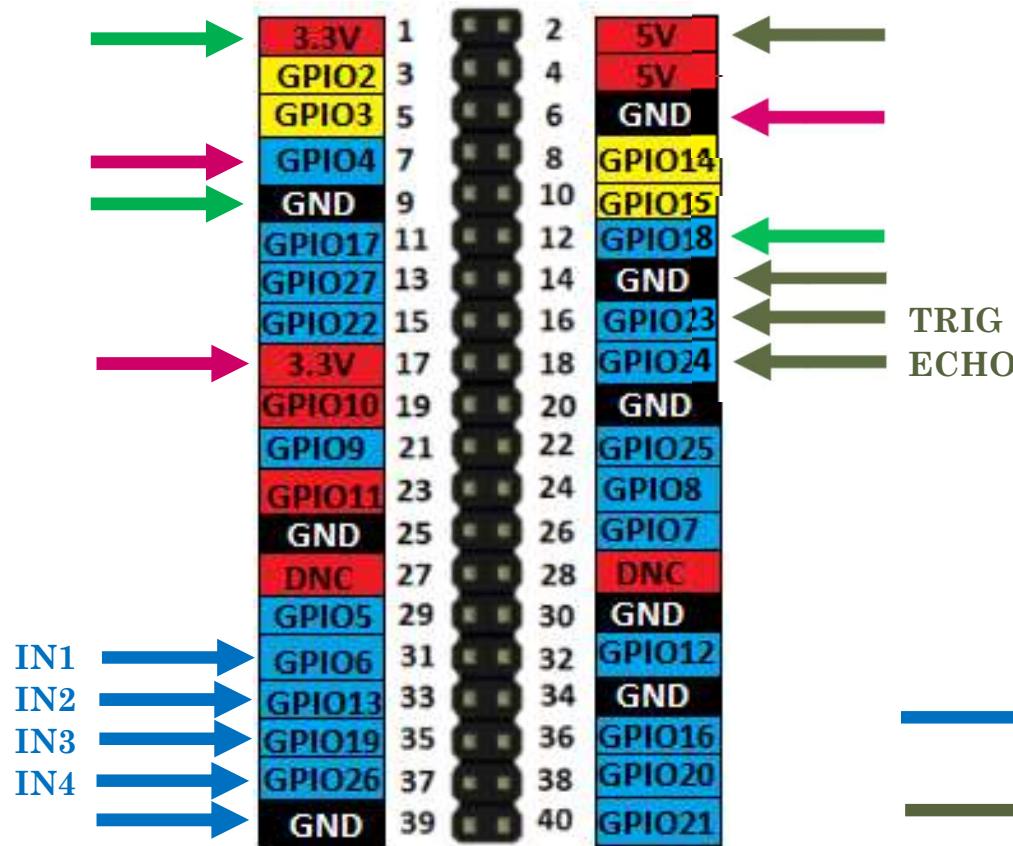
H-Bridge Circuit

- Plug **orange** female-female wire into **IN1** and pin 31
- Plug **yellow** female-female wire into **IN2** and pin 33
- Plug **green** female-female wire into **IN3** and pin 35
- Plug **blue** female-female wire into **IN4** and pin 37



Assembly

- Wire motor encoders to Raspberry Pi

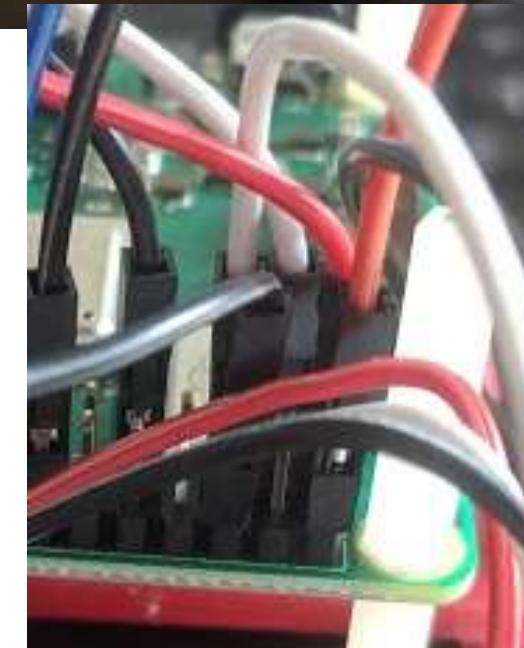
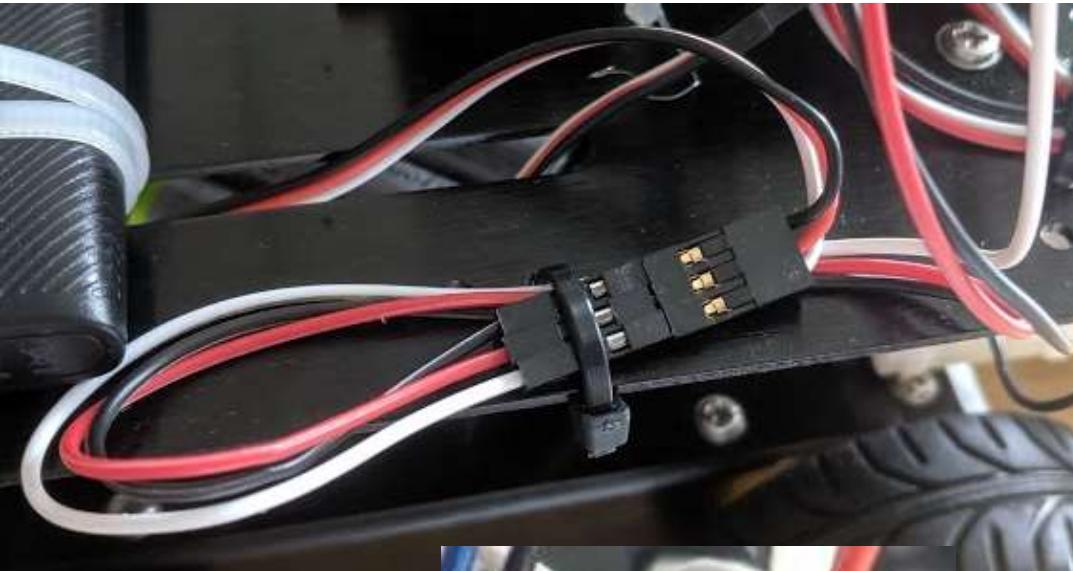


→ RPi to H-bridge
→ Distance sensor

→ Right encoder
→ Left encoder

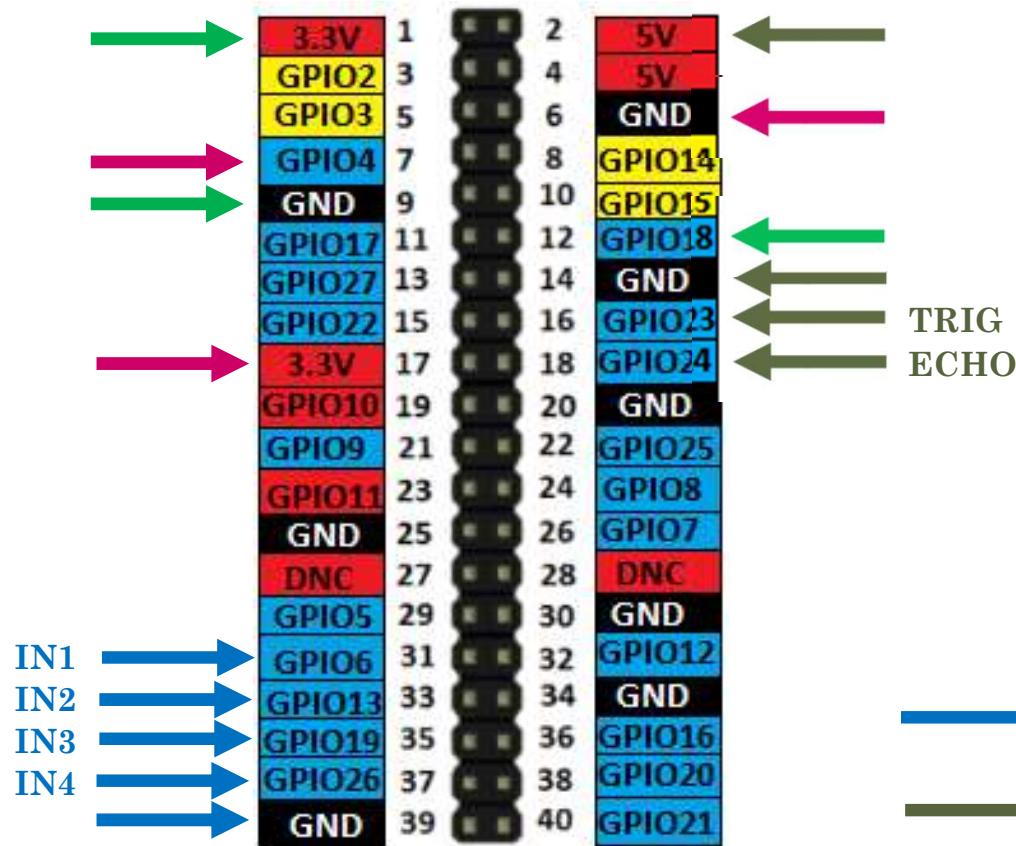
Encoder Circuit

- Plug **red** male-female wire into pin 1 or 17
 - Encoder 3.3V **power**
- Plug **black** male-female wire into pin 9 or 6
 - Encoder **GND**
- Plug white/green male-female wire into pin 12 or 7
 - Encoder signal

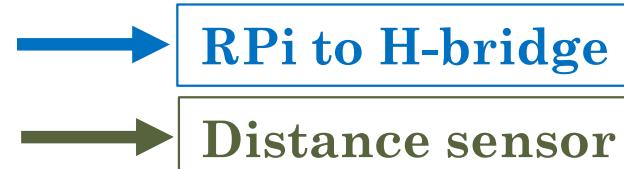
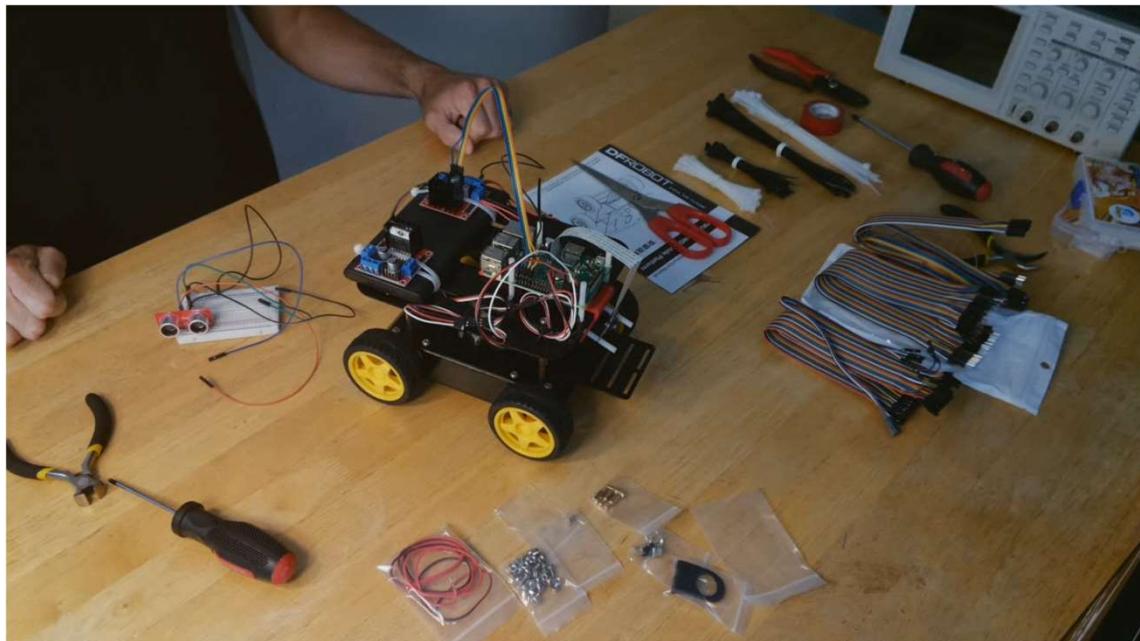


Assembly

- Mount & wire distance sensor

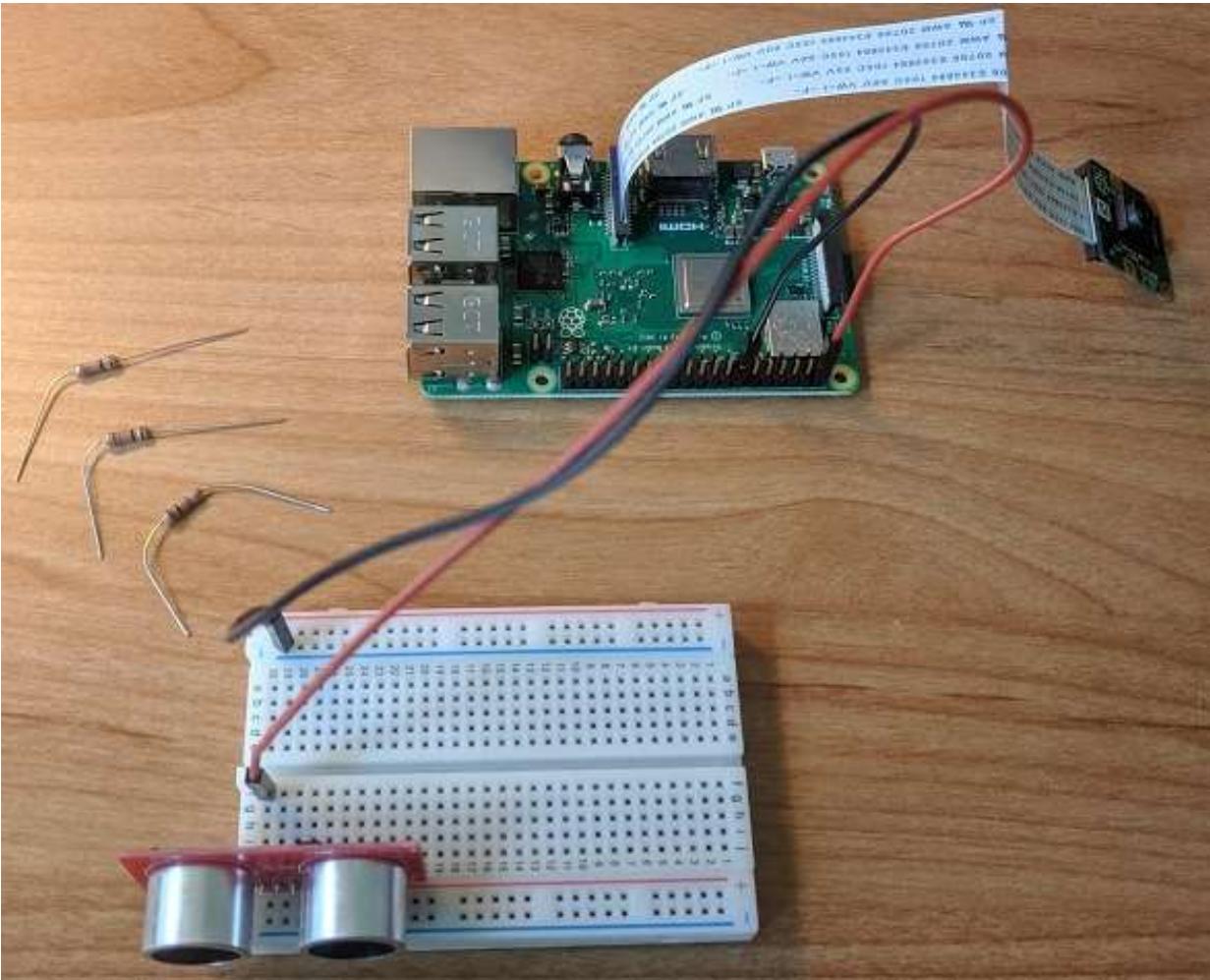


* *Do not mount breadboard via sticky tape* *



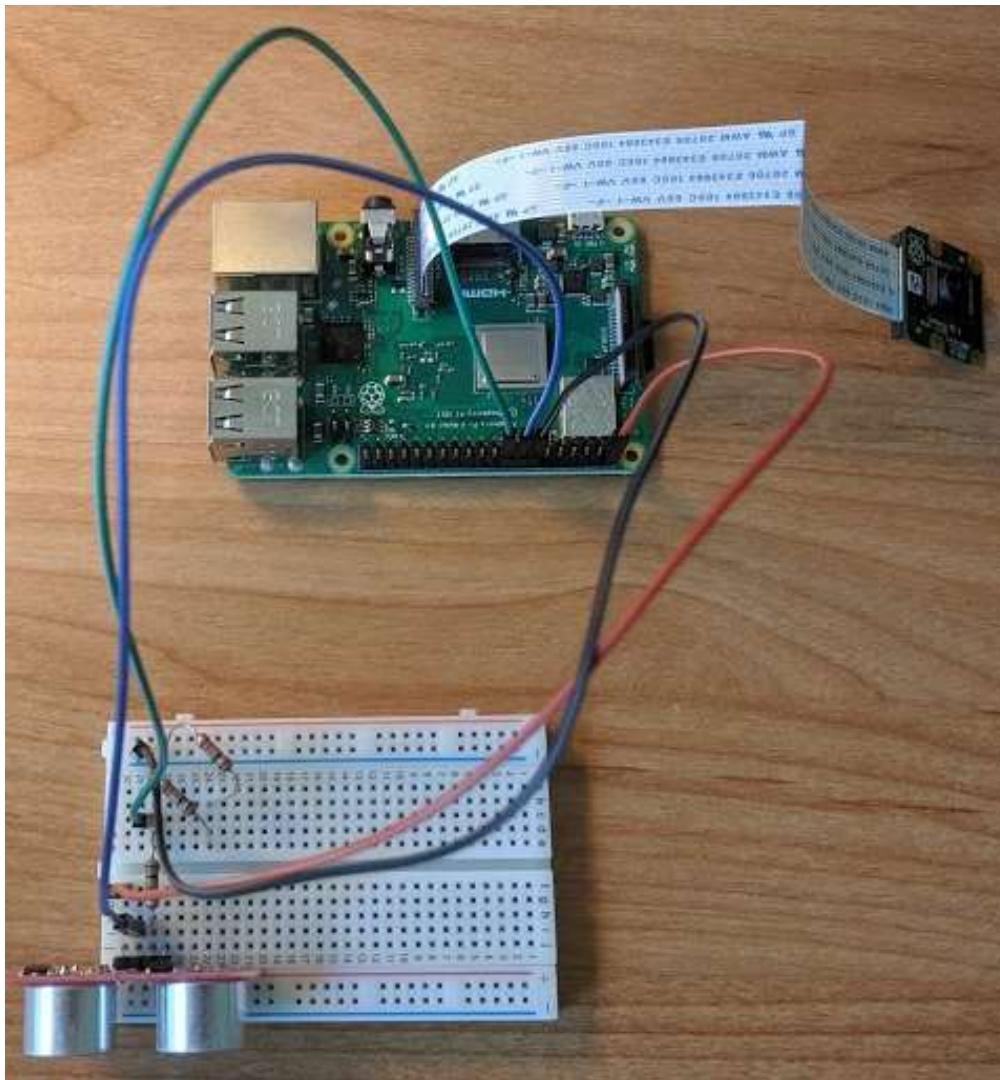
Distance Sensor Circuit

- Plug **red** male-female wire into pin 2
 - **5V** Vcc supply
- Plug **black** male-female wire into pin 14
 - GND
- Plug 5V into sensor Vcc
- Plug GND into breadboard ground rail



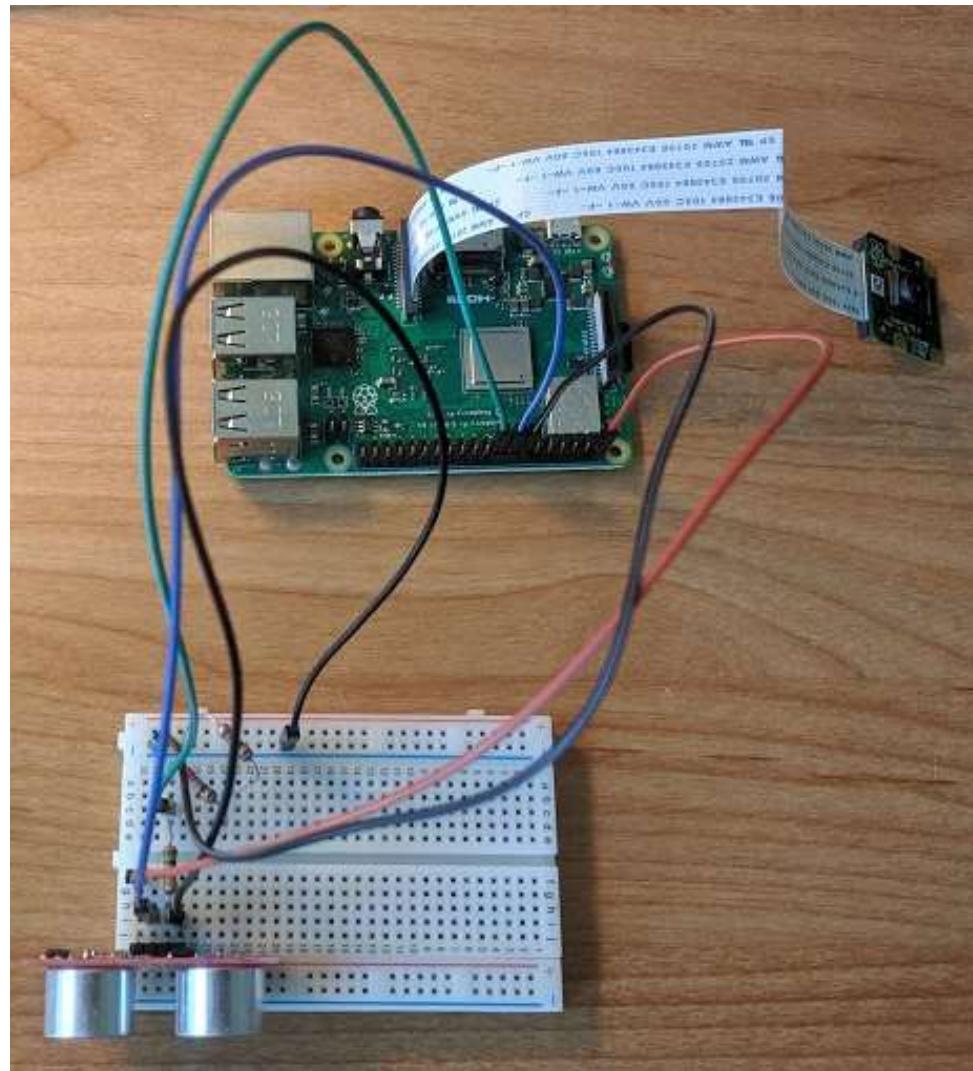
Distance Sensor Circuit

- Plug **blue** male-female wire into pin 16
 - Sensor **Trig**
- Plug **green** male-female wire into pin 18
 - Sensor **Echo**
- Connect each wire on breadboard



Distance Sensor Circuit

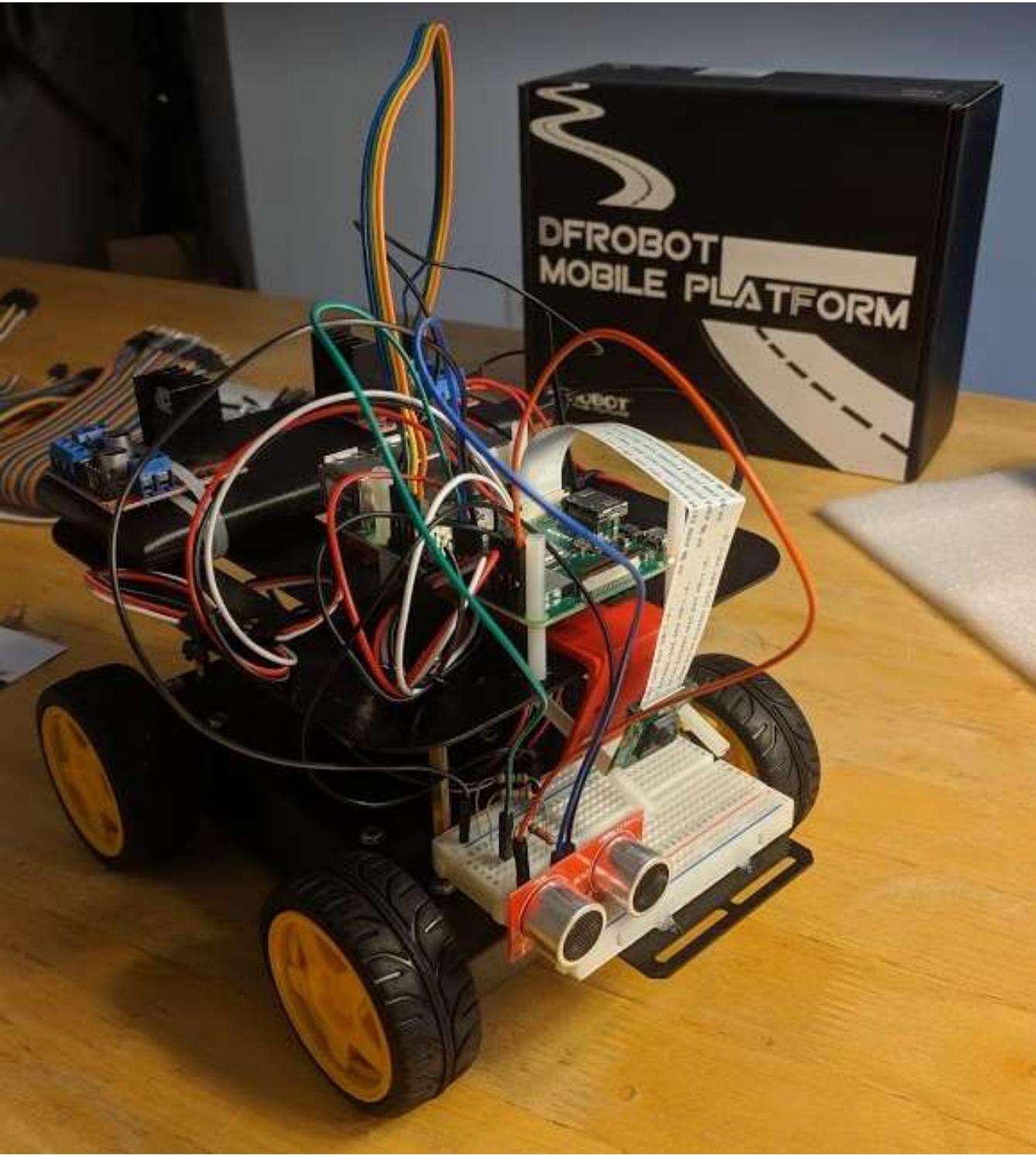
- Plug **black** male-male wire between sensor GND and breadboard GND rail



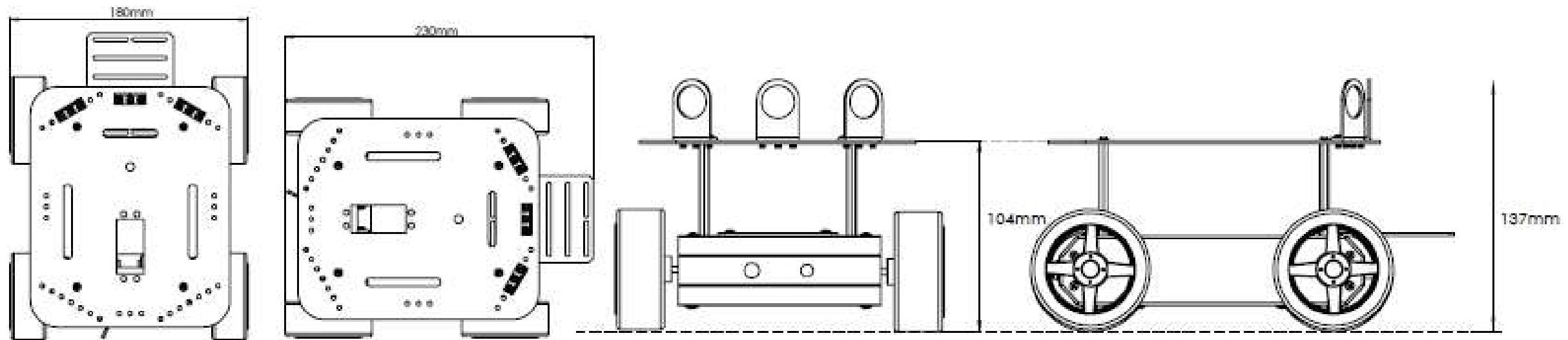
Assembly

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



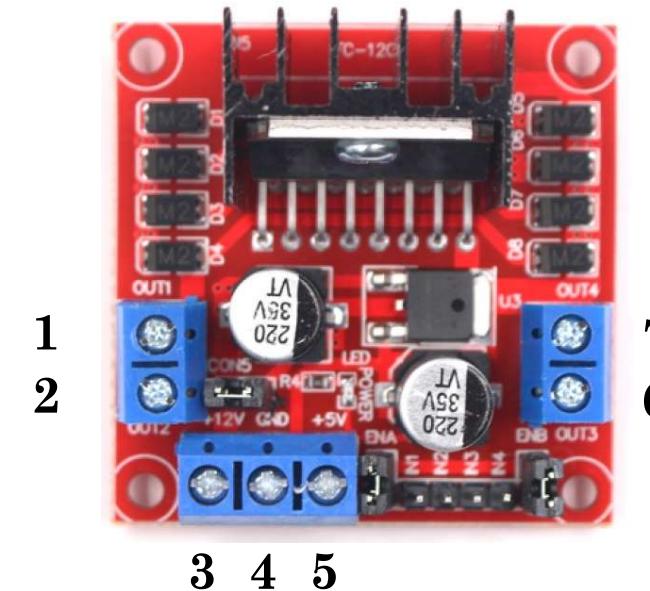
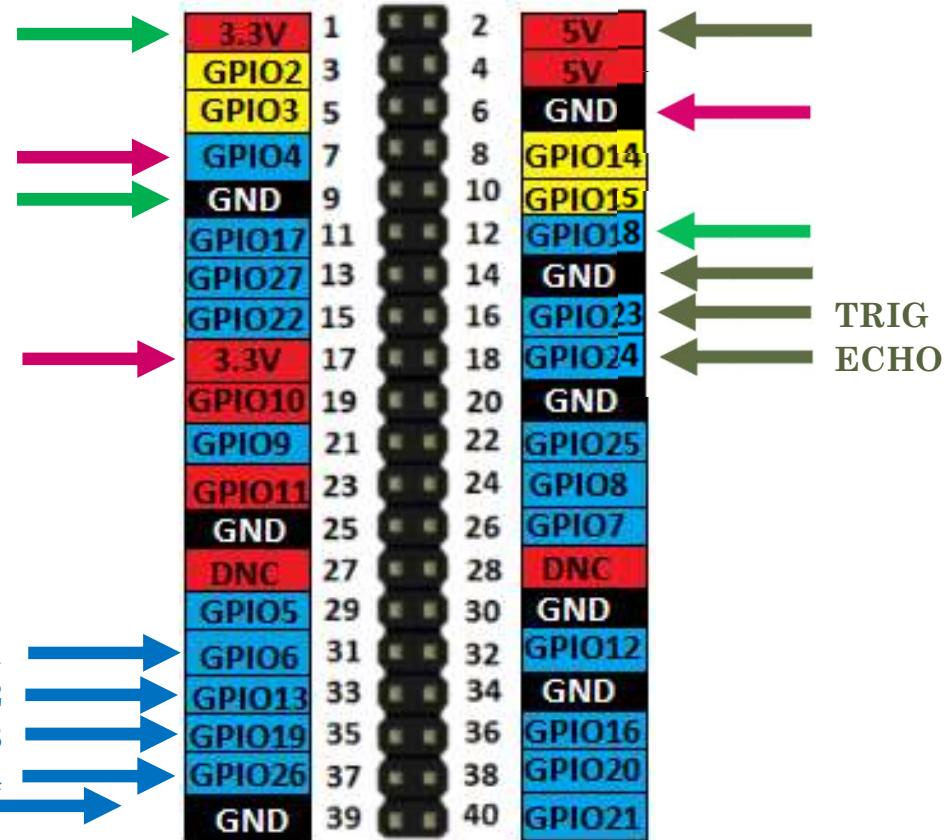


Dimensions :



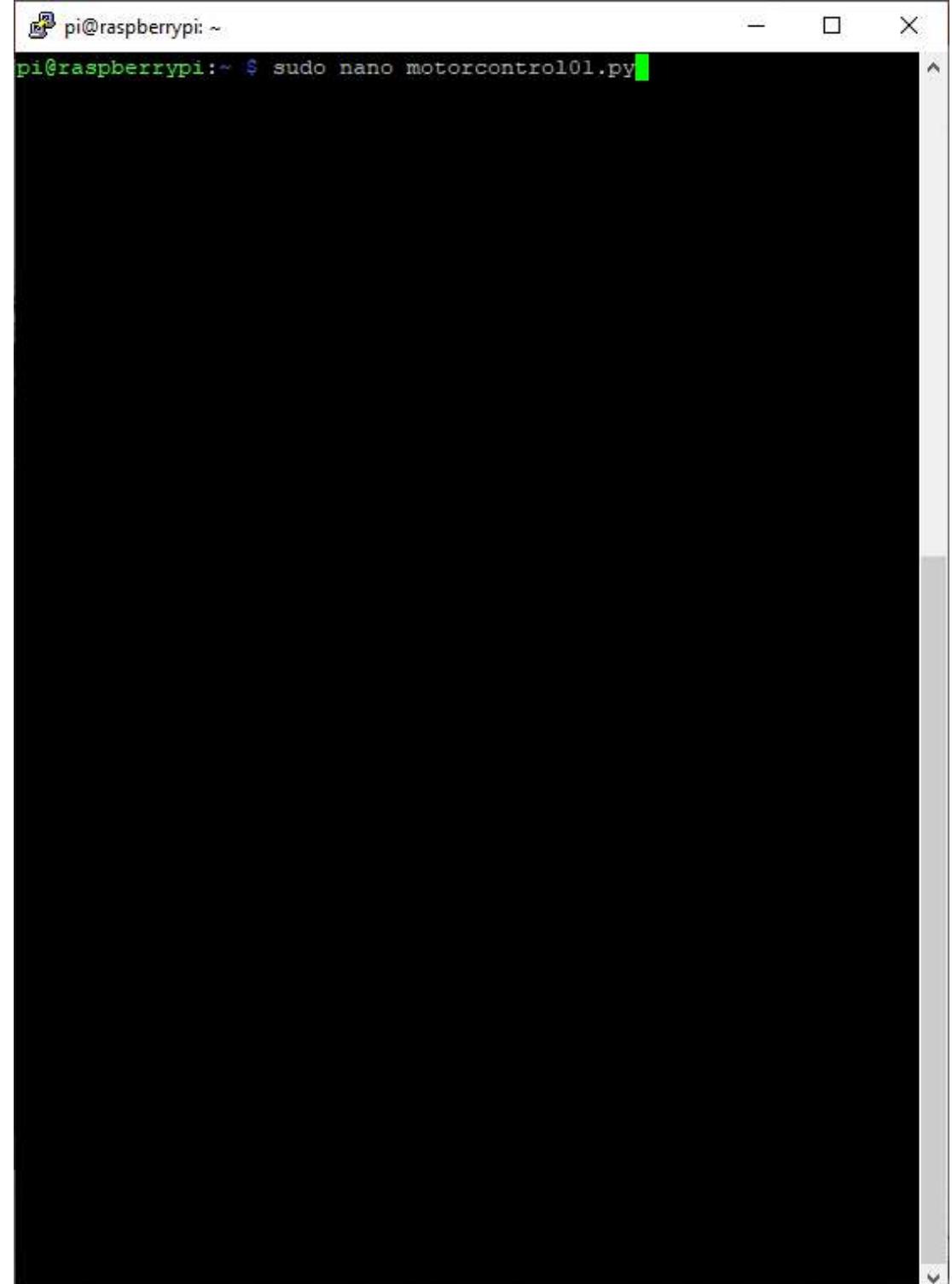
Teleoperation

- Motor direction controlled by sending a HIGH or LOW signal to the drive for each motor
- Left motors
 - IN1: HIGH & IN2: LOW
 - IN1: LOW & IN2: HIGH
- Right motors
 - IN3: HIGH & IN4: LOW
 - IN3: LOW & IN4: HIGH



Teleoperation

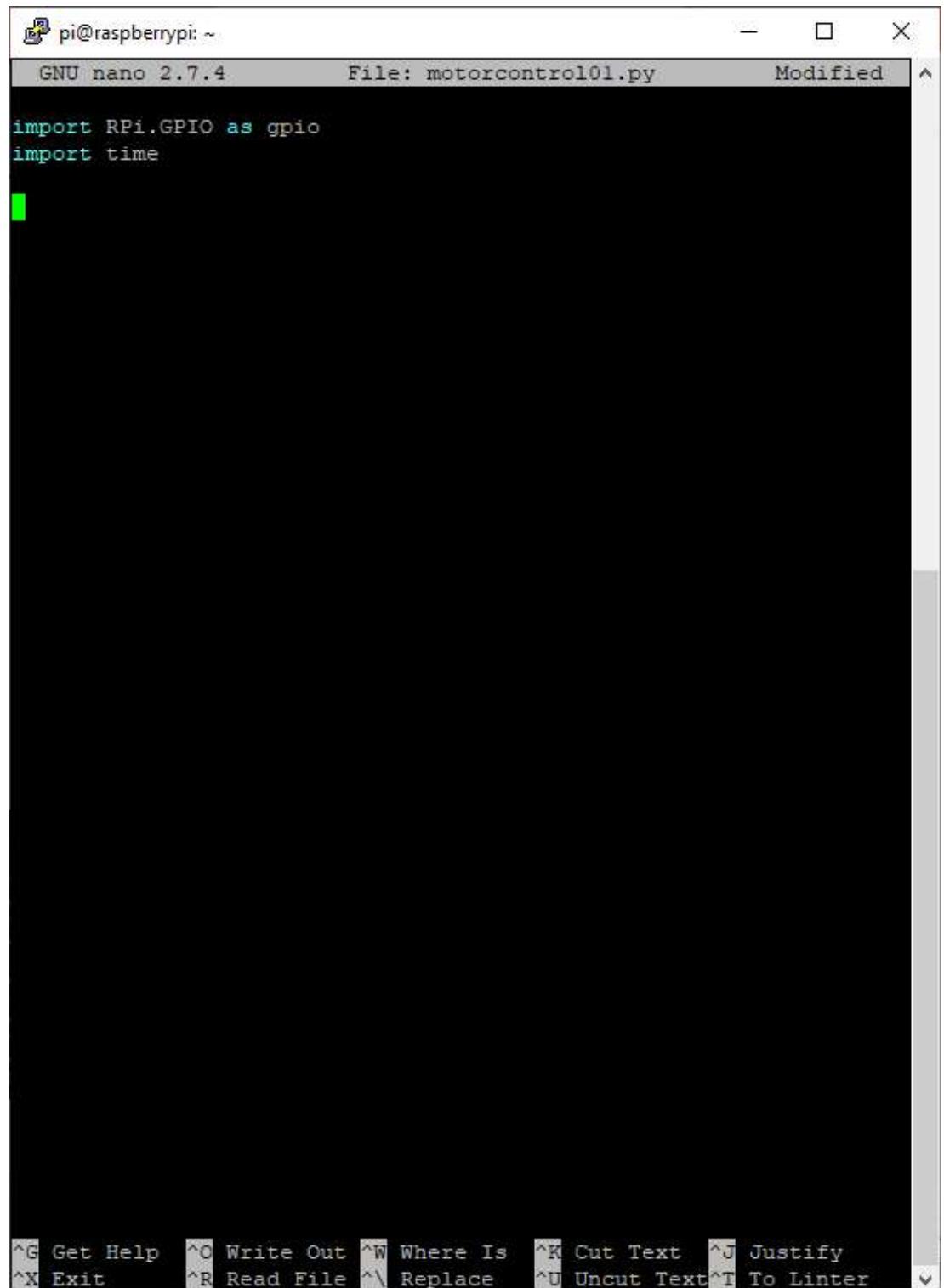
- Create new Python script:
motorcontrol01.py



A screenshot of a terminal window titled "pi@raspberrypi: ~". The window shows the command `$ sudo nano motorcontrol01.py` being typed into the text area. The terminal has a dark background with white text and a light gray scroll bar on the right side.

Teleoperation

- Import required packages



pi@raspberrypi: ~

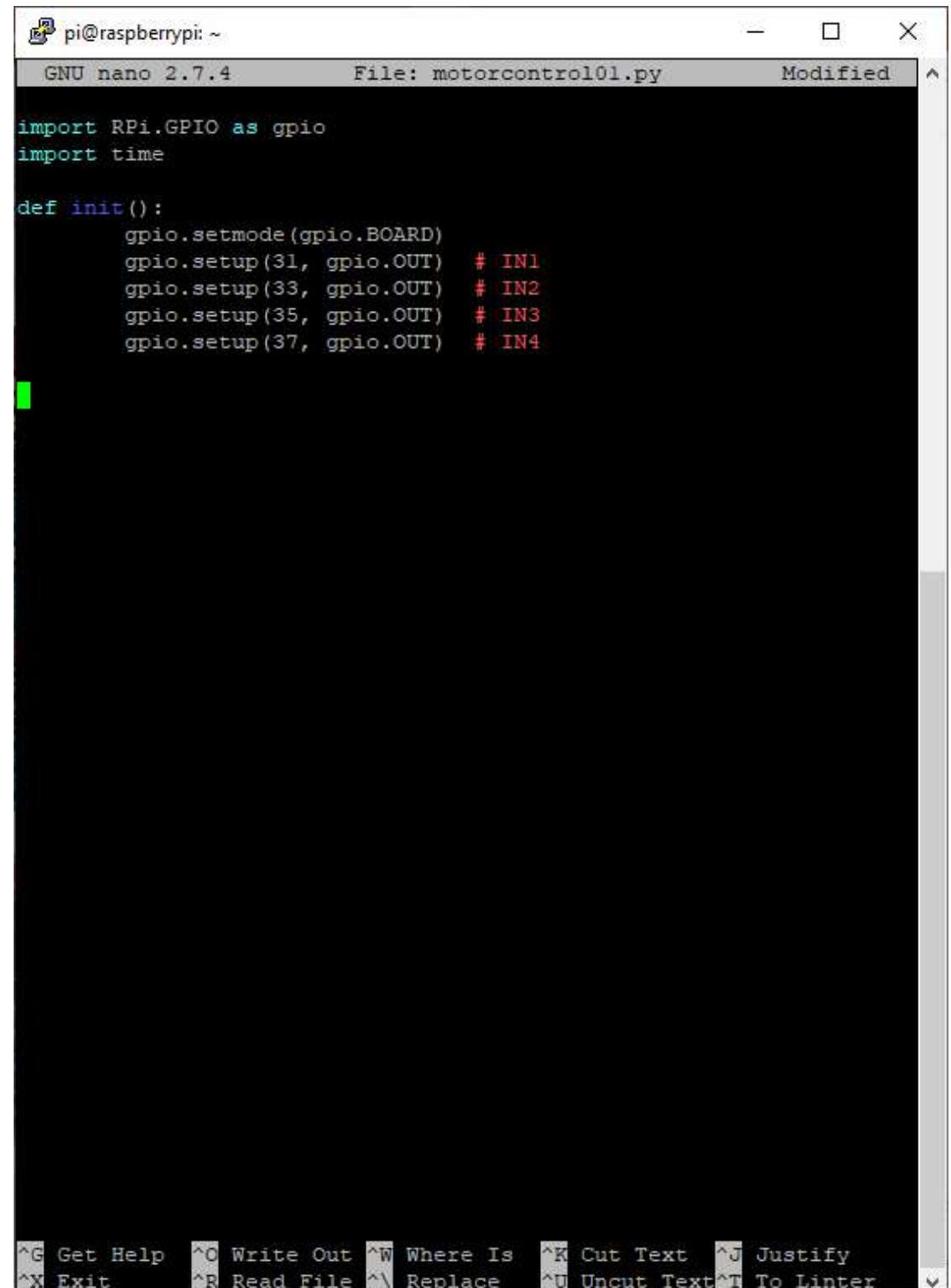
GNU nano 2.7.4 File: motorcontrol01.py Modified

```
import RPi.GPIO as gpio
import time
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Linter

Teleoperation

- Create initialization function
init()
- Set board mode
- Set pins 31 through 37 as outputs



A screenshot of a terminal window titled "pi@raspberrypi: ~". The window shows a file named "motorcontrol01.py" being edited with "GNU nano 2.7.4". The code in the file is as follows:

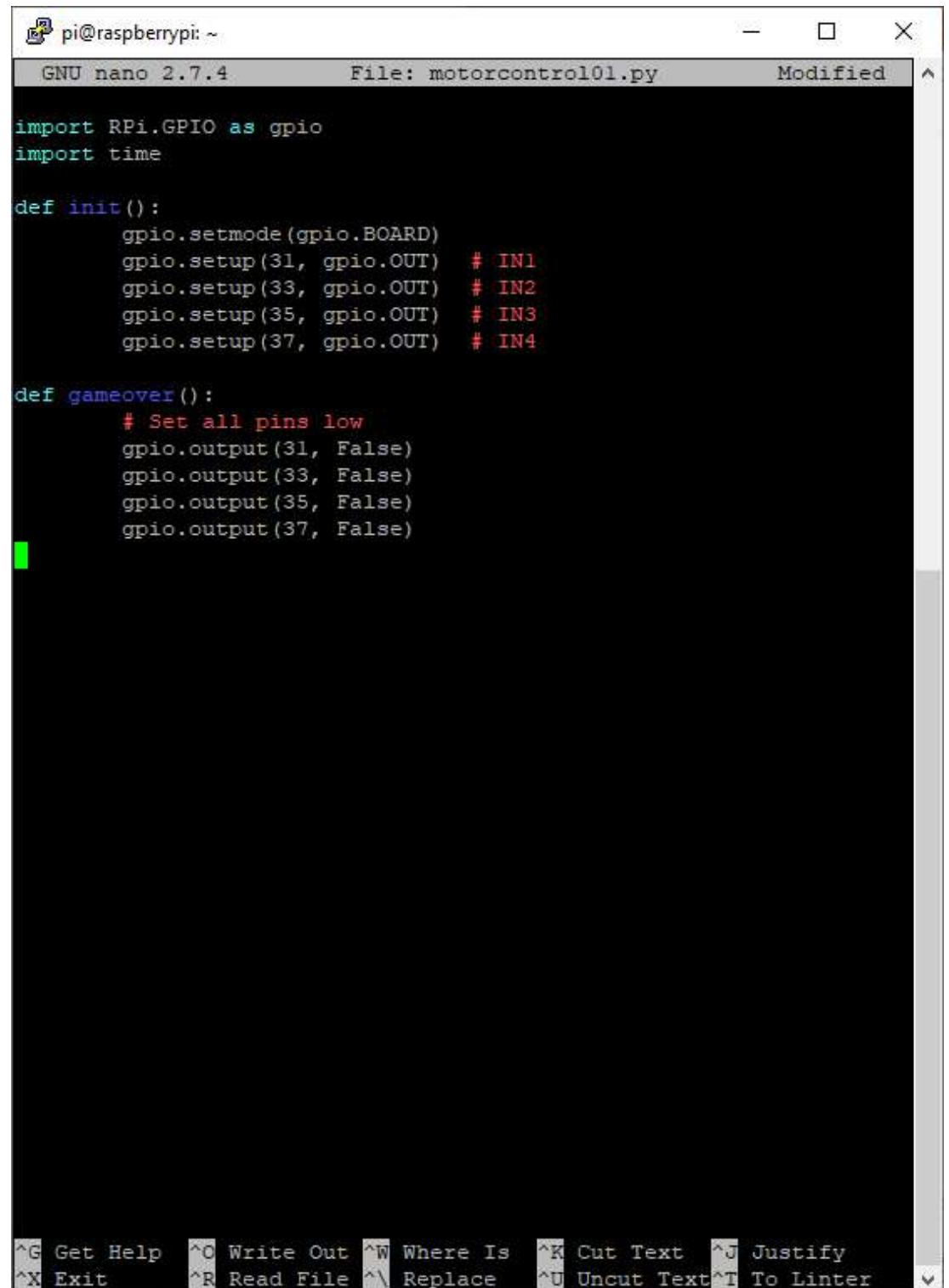
```
import RPi.GPIO as gpio
import time

def init():
    gpio.setmode(gpio.BOARD)
    gpio.setup(31, gpio.OUT) # IN1
    gpio.setup(33, gpio.OUT) # IN2
    gpio.setup(35, gpio.OUT) # IN3
    gpio.setup(37, gpio.OUT) # IN4
```

The terminal window has a dark background with light-colored text. A green vertical bar is visible on the left side of the screen. At the bottom of the window, there is a menu bar with various keyboard shortcut options.

Teleoperation

- Define *gameover()* function
- Set all pins low



The screenshot shows a terminal window titled "pi@raspberrypi: ~" running "GNU nano 2.7.4". The file being edited is "motorcontrol01.py", which contains the following Python code:

```
import RPi.GPIO as gpio
import time

def init():
    gpio.setmode(gpio.BOARD)
    gpio.setup(31, gpio.OUT) # IN1
    gpio.setup(33, gpio.OUT) # IN2
    gpio.setup(35, gpio.OUT) # IN3
    gpio.setup(37, gpio.OUT) # IN4

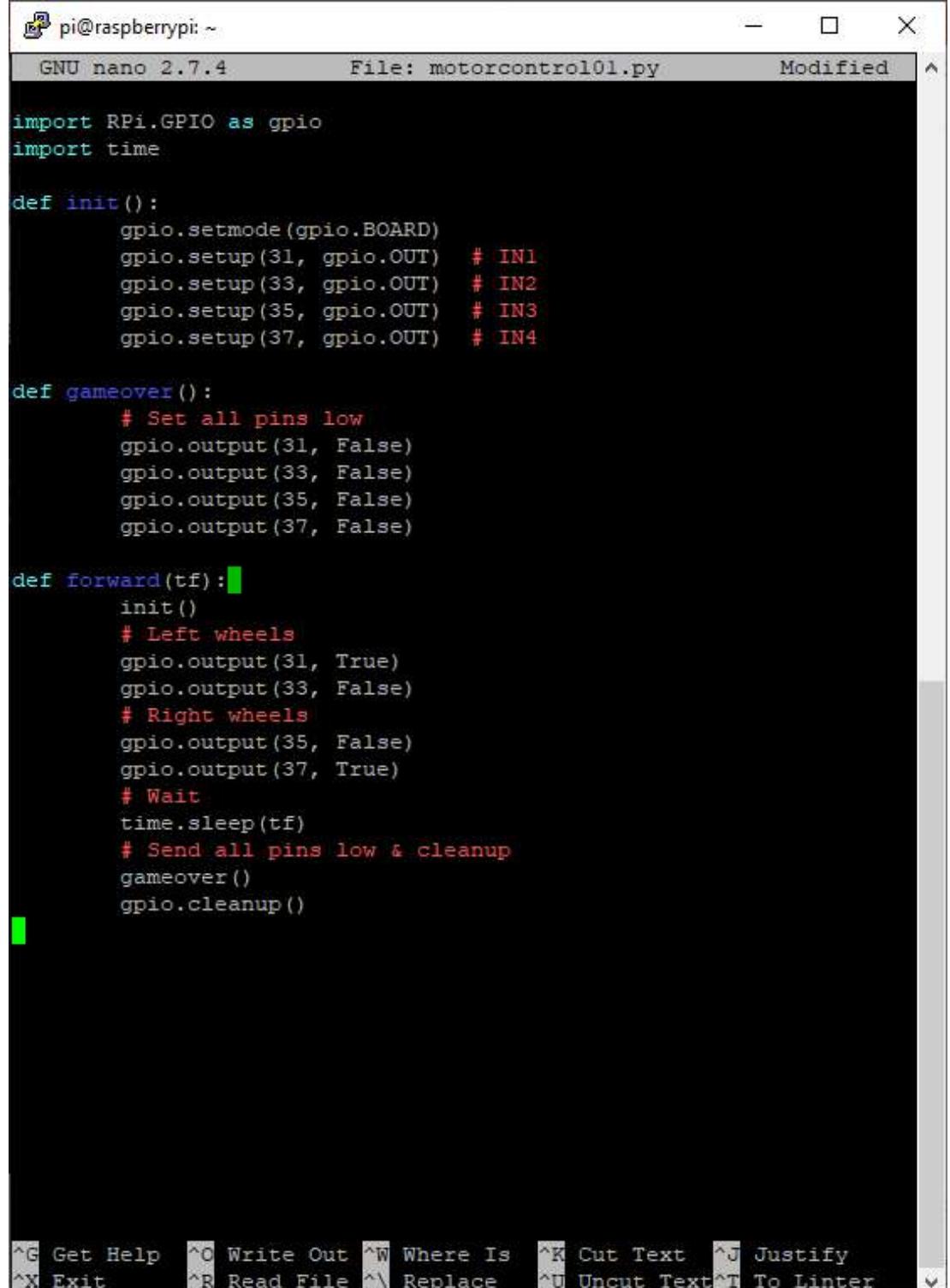
def gameover():
    # Set all pins low
    gpio.output(31, False)
    gpio.output(33, False)
    gpio.output(35, False)
    gpio.output(37, False)
```

At the bottom of the terminal window, there is a menu bar with the following options:

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Linter

Teleoperation

- Define *forward()* function
- Function takes as an input the length of time **tf** to run function
- Once complete, set all pins low & clean up gpio



The screenshot shows a terminal window titled "pi@raspberrypi: ~" running "GNU nano 2.7.4". The file being edited is "motorcontrol01.py", which contains the following Python code:

```
import RPi.GPIO as gpio
import time

def init():
    gpio.setmode(gpio.BOARD)
    gpio.setup(31, gpio.OUT) # IN1
    gpio.setup(33, gpio.OUT) # IN2
    gpio.setup(35, gpio.OUT) # IN3
    gpio.setup(37, gpio.OUT) # IN4

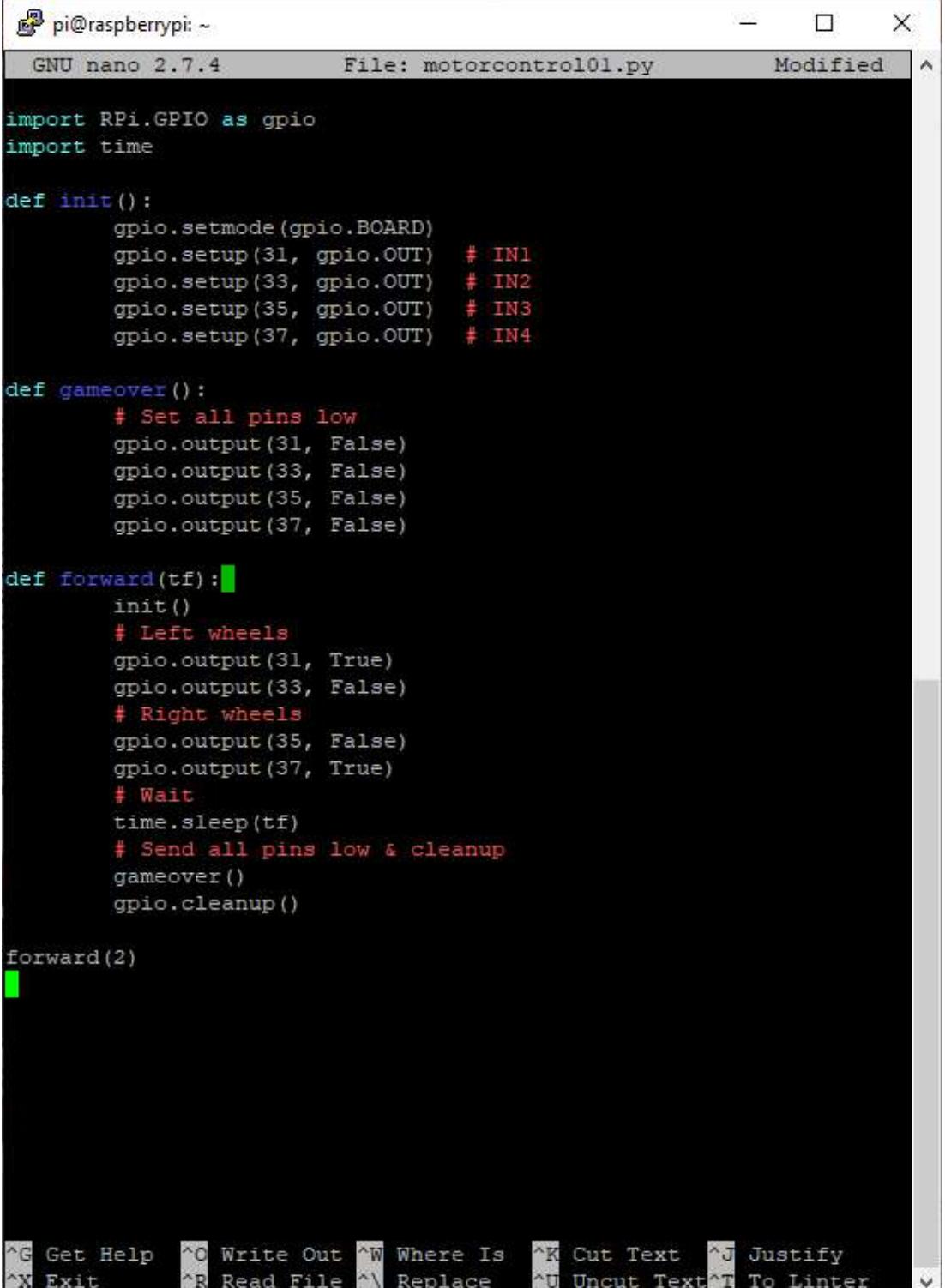
def gameover():
    # Set all pins low
    gpio.output(31, False)
    gpio.output(33, False)
    gpio.output(35, False)
    gpio.output(37, False)

def forward(tf):
    init()
    # Left wheels
    gpio.output(31, True)
    gpio.output(33, False)
    # Right wheels
    gpio.output(35, False)
    gpio.output(37, True)
    # Wait
    time.sleep(tf)
    # Send all pins low & cleanup
    gameover()
    gpio.cleanup()

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text ^T To Linter
```

Teleoperation

- Call *forward()* function
- Requires time input in seconds



The screenshot shows a terminal window titled "pi@raspberrypi: ~" running "GNU nano 2.7.4". The file being edited is "motorcontrol01.py", which has been modified. The code is as follows:

```
import RPi.GPIO as gpio
import time

def init():
    gpio.setmode(gpio.BOARD)
    gpio.setup(31, gpio.OUT) # IN1
    gpio.setup(33, gpio.OUT) # IN2
    gpio.setup(35, gpio.OUT) # IN3
    gpio.setup(37, gpio.OUT) # IN4

def gameover():
    # Set all pins low
    gpio.output(31, False)
    gpio.output(33, False)
    gpio.output(35, False)
    gpio.output(37, False)

def forward(tf):
    init()
    # Left wheels
    gpio.output(31, True)
    gpio.output(33, False)
    # Right wheels
    gpio.output(35, False)
    gpio.output(37, True)
    # Wait
    time.sleep(tf)
    # Send all pins low & cleanup
    gameover()
    gpio.cleanup()

forward(2)
```

The terminal window includes standard nano key bindings at the bottom:

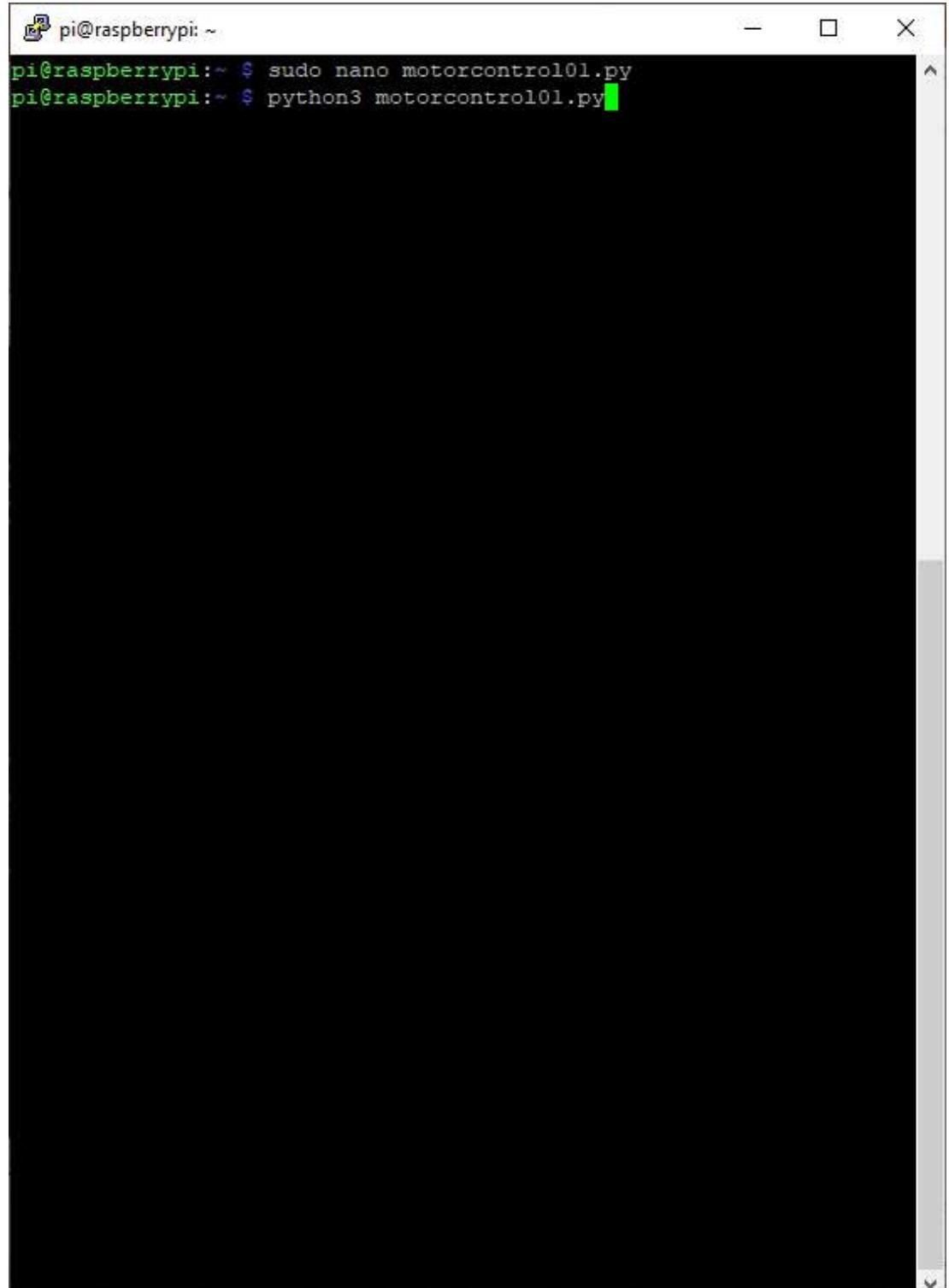
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Linter

Teleoperation

- **Mount robot on top of box**
such that wheels can spin freely
- Run program

If error such as “channel sent is invalid” run:
`pip3 install --upgrade --pre rpi.gpio`

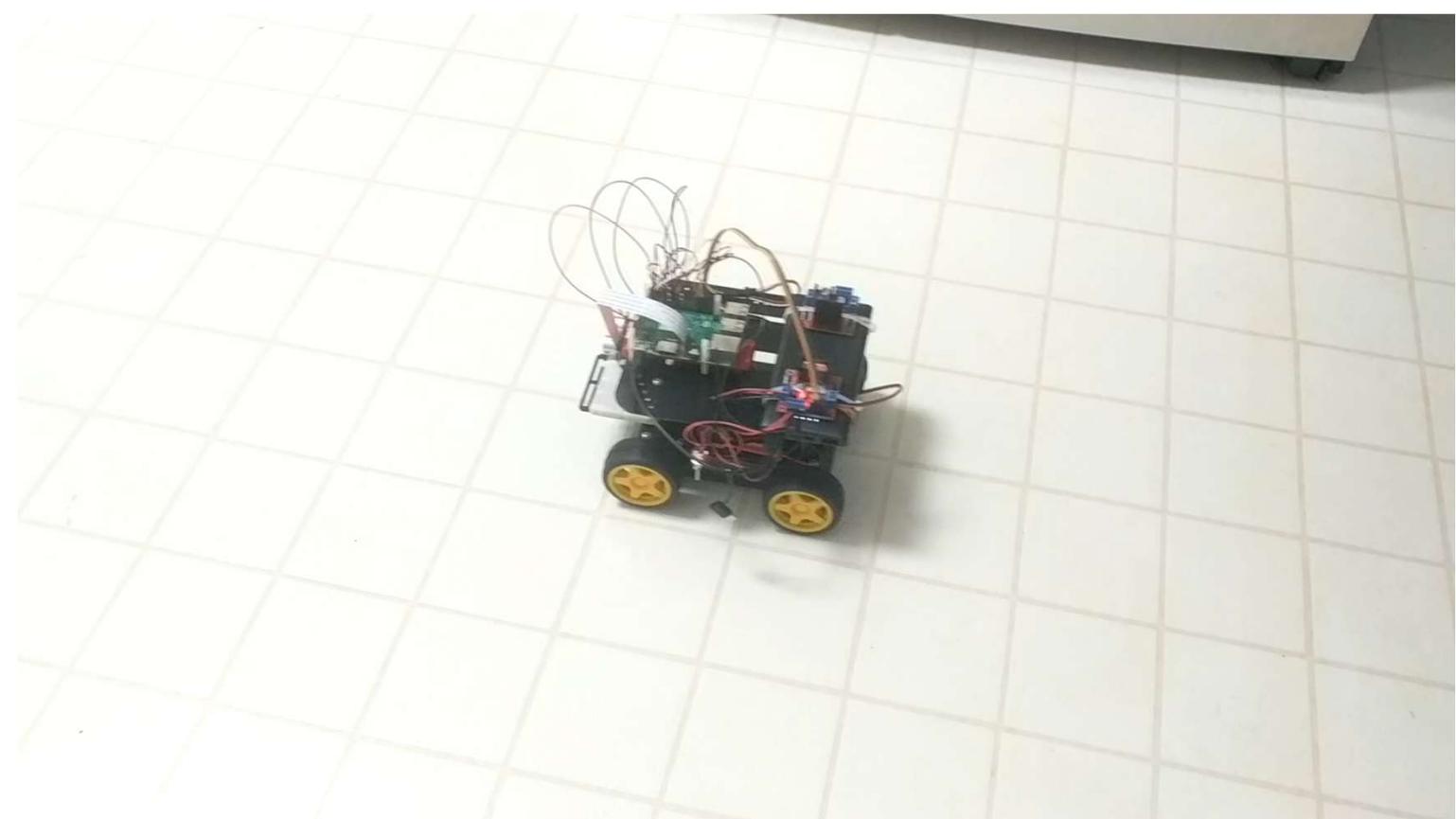
(both are double-dashes)



A screenshot of a terminal window titled "pi@raspberrypi: ~". The window shows two commands being run: "sudo nano motorcontrol01.py" and "python3 motorcontrol01.py". The second command is currently being typed, with the cursor at the end of "motorcontrol01.py". The terminal has a dark background with light-colored text.

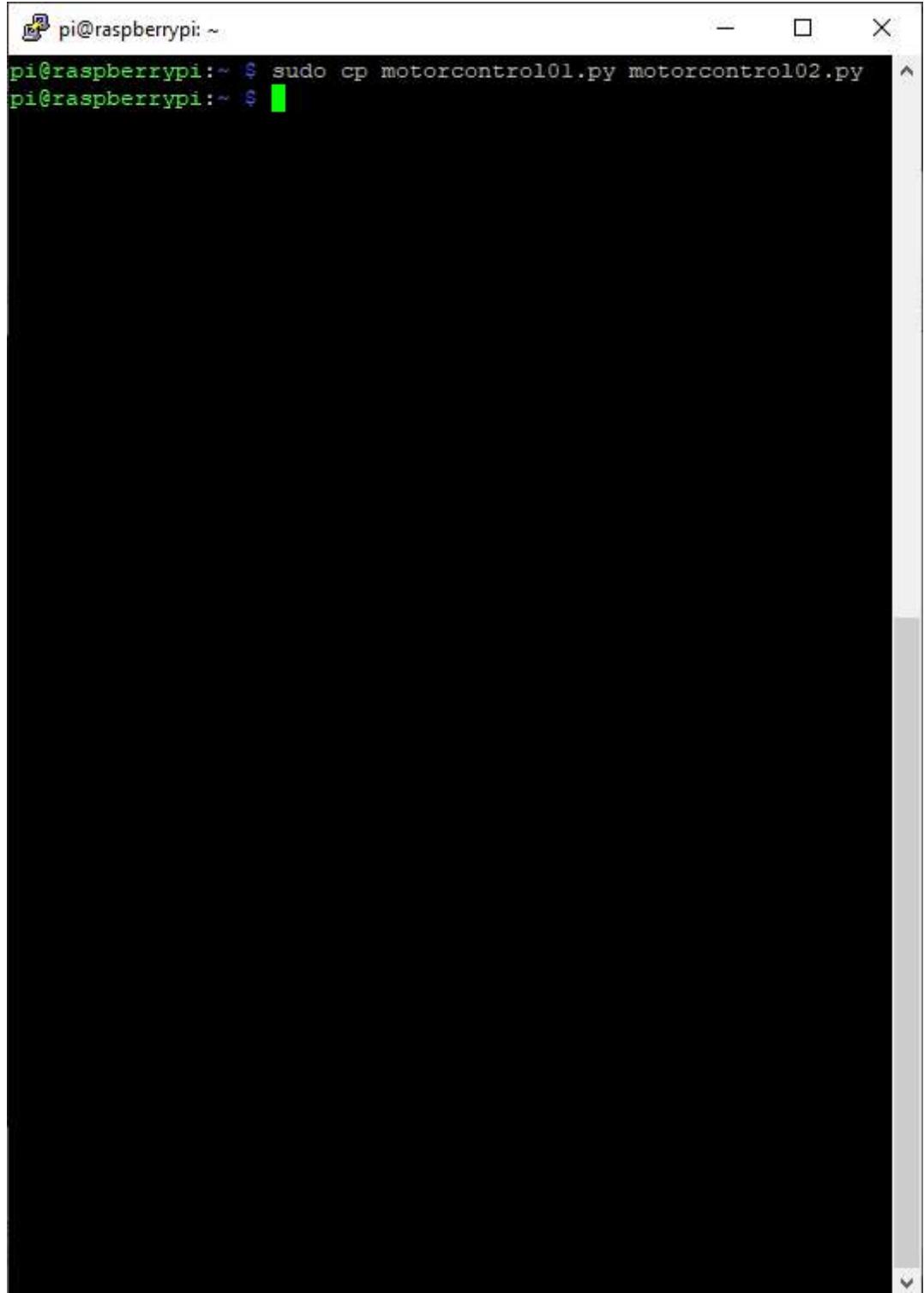
In-Class Exercise

- Update the motorcontrol01.py script to:
 1. Drive forward
 2. Drive in reverse
 3. Pivot left
 4. Pivot right



Teleoperation

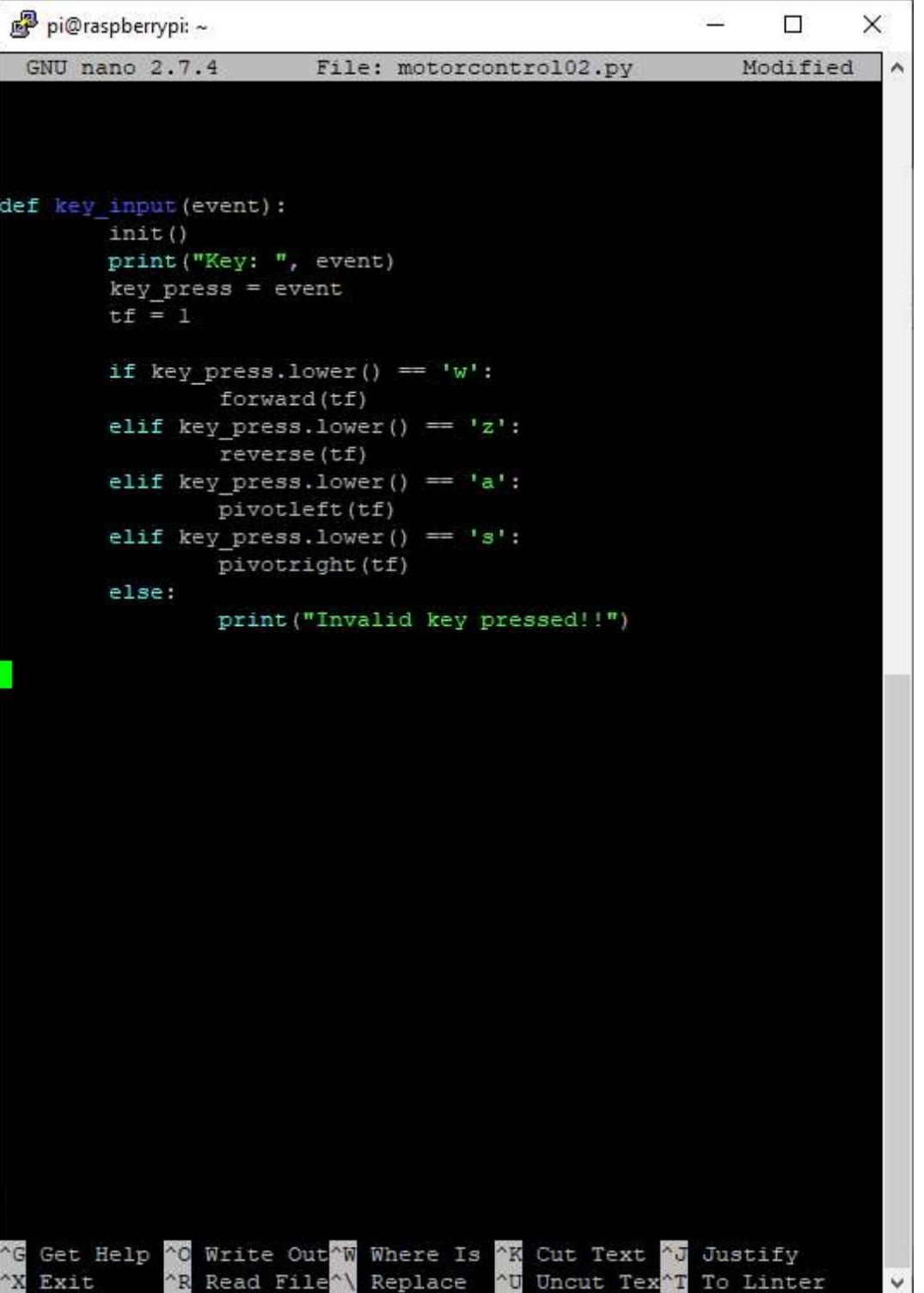
- Copy Python script over to new file: ***motorcontrol02.py***



```
pi@raspberrypi: ~
pi@raspberrypi:~ $ sudo cp motorcontrol01.py motorcontrol02.py
pi@raspberrypi:~ $
```

Teleoperation

- After forward(), reverse(), pivotleft(), and pivotright() functions...
- Define *key_input()* function
- Takes user-defined input from keyboard
- Assigns driving direction based on user input



```
pi@raspberrypi: ~
GNU nano 2.7.4      File: motorcontrol02.py      Modified
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!!")
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Uncut Tex ^T To Linter

Teleoperation

- Setup while True loop

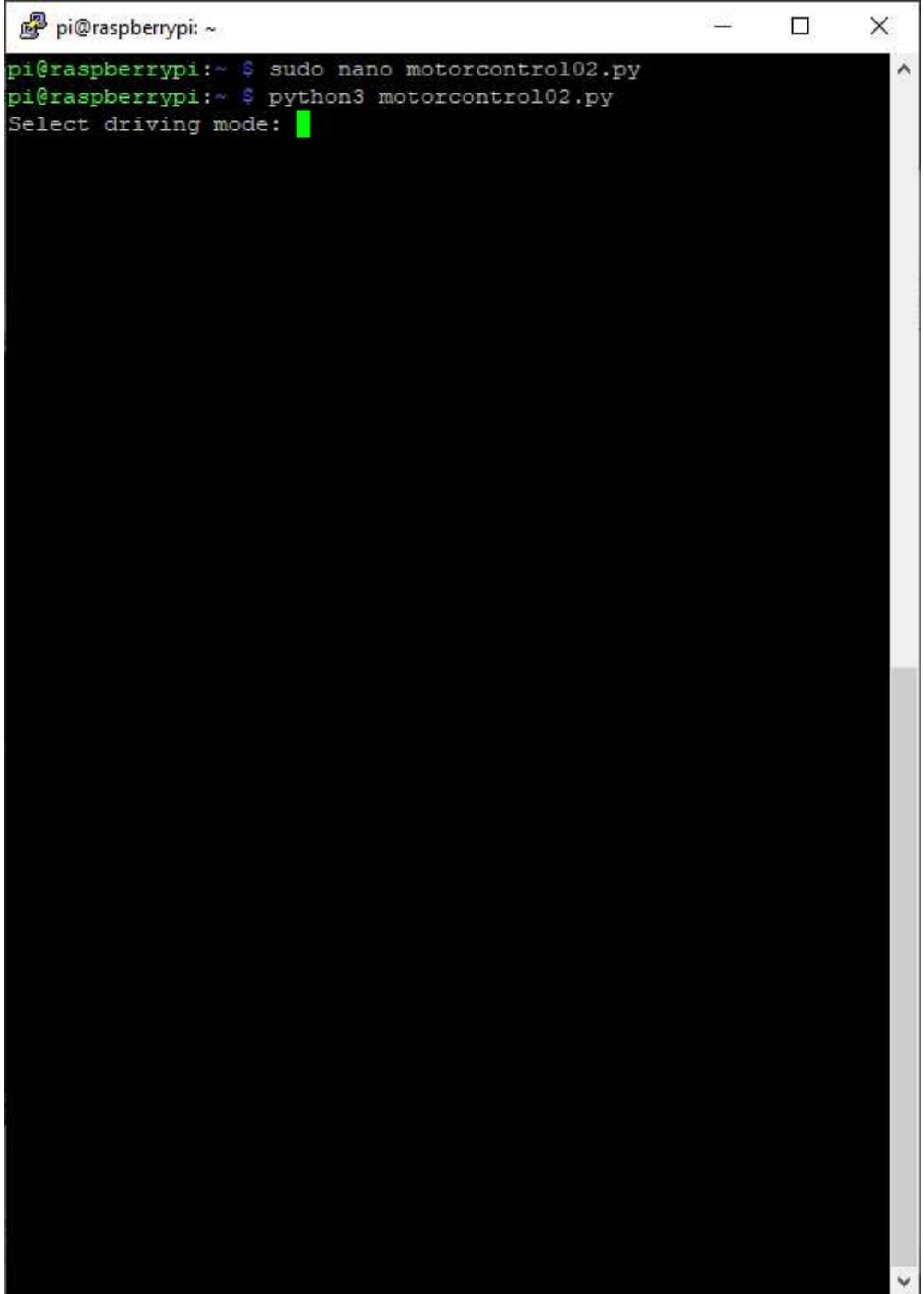
```
def key_input(event):
    init()
    print("Key: ", event)
    key_press = event
    tf = 1

    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:
    key_press = input("Select driving mode: ")
    if key_press == 'p':
        break
    key_input(key_press)
```

Teleoperation

- **Mount robot on top of box**
such that wheels can spin
freely
- Run program



A terminal window titled "pi@raspberrypi: ~" showing a command-line interface for motor control. The window contains the following text:

```
pi@raspberrypi:~ $ sudo nano motorcontrol02.py
pi@raspberrypi:~ $ python3 motorcontrol02.py
Select driving mode: █
```

The terminal window has standard Linux-style window controls at the top right.

References

- *Introduction to Autonomous Mobile Robots*, Siegwart
 - Chapter 2, 3
- *Introduction to Robotics*, Asada
 - Chapter 2
 - <https://ocw.mit.edu/courses/mechanical-engineering/2-12-introduction-to-robotics-fall-2005/lecture-notes/chapter2.pdf>