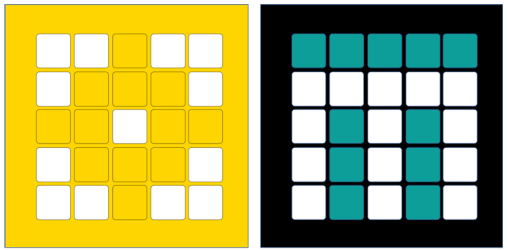
PRIME LESSONS

By the Makers of EV3Lessons



MORE ACCURATE TURNS

BY SANJAY AND ARVIND SESHAN

This lesson uses SPIKE 3 software

LESSON OBJECTIVES

Learn how to improve the accuracy of turns

Learn alternative ways to do pivot and spin turns

HOW ACCURATE IS YOUR TURN?

Run this code and use the Dashboard to see if turning 90 degrees actually turns 90 degrees.

```
from hub import port, motion_sensor
import runloop, motor_pair, sys

# Function that returns true when the absolute yaw angle is 90 degrees
def turn_done():
    # convert tuple decidegree into same format as in app and blocks
    return abs(motion_sensor.tilt_angles()[0] * -0.1) > 90

async def main():
    motor_pair.pair(motor_pair.PAIR_1, port.C, port.D)
    motion_sensor.reset_yaw(0)
    await runloop.until(motion_sensor.stable)
    # Use a steering of 100 instead of 50 to do a spin turn
    motor_pair.move(motor_pair.PAIR_1, 50, velocity=500)
    await runloop.until(turn_done)
    motor_pair.stop(motor_pair.PAIR_1)
    sys.exit("Done")

runloop.run(main())
```

HOW ACCURATE IS YOUR TURN?

Note that we have set the motor speed to 500 instead of 200 in the previous lesson.

For Drive Base 1, it turns 98 degrees

This is for two reasons

1. It takes a short time to read the gyro. In this time, the robot has moved. This delay on the SPIKE Prime is relatively small but will produce a few degrees of error.
2. It takes some time to stop the robot since it has momentum. This produces several degrees of additional error.

IMPROVING TURN ACCURACY

As we mentioned on the previous slide, using Drive base 1 at 500 velocity, it turns 98 degrees instead of the requested 90 degrees. How do we solve this problem?

One solution is to ask it to turn 8 degrees less for Drive Base 1.

> The amount to reduce your turn will depend on the speed of your turn and your robot's physical design. You will need to try some values to get this right.

A better solution is to use a lower speed. With a speed of 200, the error for Drive Base 1 dropped from 8 degrees to just 2 degrees.

We did not notice any significant difference using move vs move_tank. Adjusting the speed made the biggest difference.

Pivot turns and spin turns have a similar error pattern

TURNING WITHOUT USING GYRO

It is possible to compute the degrees the wheel has to turn to turn the robot by a certain amount, using the robot geometry

Remember, motor turns are not the same as robot turns!

Using Drive Base 1 (DB1) as an example:

First, we need to know the wheel circumference. For DB1, that is 17.5cm

Next, we need to know the distance between the wheels. This is known as the TRACK. You can measure it. For DB1, this is approximately 11.2 cm.

For DB1, the motors are directly driving the wheels without gears. So, we know that there is no additional gear ratio multiplier (i.e., it is 1)

Now, we can calculate the math to do spin and pivot turns

GEOMETRIC SPIN TURN

A spin turn turns the robot around the center of its drive wheels.

The **diameter** of this spin turn circle is equal to the track. Its circumference is given by: Cs = Track * Pi

To make one full turn (360 degrees), each wheel has to move a distance equal to the circumference of this turn circle.

One motor rotation = one wheel rotation

If the circumference of the wheel is given by Cw, then:

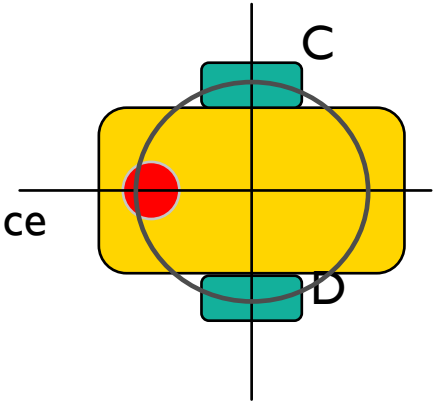
motor **rotations** for spinning the robot 360 degrees = (Cs /Cw)

motor **degrees** for spinning the robot 360 degrees = (Cs /Cw) * 360

In general terms, for turning "robot_turn" degrees:

motor_degrees = (Cs/Cw) * robot_turn

This works for **any** value of degrees. There is no special case for crossing 180, 360 or any other value, as we saw when using the yaw angle.



CHALLENGE - SPIN

Write a function to make any spin turn, clockwise or counterclockwise, using just MotorPair functions.

Use a parameter to make the speed variable

Write tests for your function

CHALLENGE – SPIN: SOLUTION PAGE 1 OF 2

```
from hub import port
import motor_pair, runloop, sys, math

# Constants for Drive Base 1
motor_pair.pair(motor_pair.PAIR_1, port.C, port.D)
WHEEL_CIRCUMFERENCE = 17.5 # cm - please adjust according to your robot wheel
TRACK = 11.2 # cm - please measure your own robot.
SPIN_CIRCUMFERENCE = TRACK * math.pi

async def spin_turn(robot_degrees, motor_speed):
    # Add a multiplier for gear ratios if you're using gears
    motor_degrees = int((SPIN_CIRCUMFERENCE/WHEEL_CIRCUMFERENCE) * abs(robot_degrees))
    if robot_degrees > 0:
        # spin clockwise
        await motor_pair.move_for_degrees(motor_pair.PAIR_1, motor_degrees, 100, velocity=motor_speed)
    else:
        #spin counter clockwise
        await motor_pair.move_for_degrees(motor_pair.PAIR_1, motor_degrees, -100, velocity=motor_speed)
```

Define the imports, the globals that are fixed for your robot, and the spin_turn function that computes the motor move degrees.

CHALLENGE – SPIN: SOLUTION PAGE 2 OF 2

```
async def main():
    # Spin 720 clockwise with speed 200
    await spin_turn(720, 200)
    await runloop.sleep_ms(1000)
    # Spin 180 counterclockwise with speed 200
    await spin_turn(-180, 200)
    await runloop.sleep_ms(1000)
    # Spin 90 clockwise with speed 300
    await spin_turn(90, 300)
    # stop and exit the program
    sys.exit("Stopping")

runloop.run(main())
```

Write your tests in the main function. When you are satisfied that your code works, the function can be added to your library if you choose to use geometric rotation

GEOMETRIC PIVOT TURN

A pivot turn turns the robot around the center of **one** of its drive wheels.

The **radius** of this pivot turn circle is equal to the track. Its circumference is given by: Cp = 2 * Track * Pi

To make one full turn (360 degrees), each wheel has to move a distance equal to the circumference of this turn circle.

One motor rotation = one wheel rotation

If the circumference of the wheel is given by Cw, then:

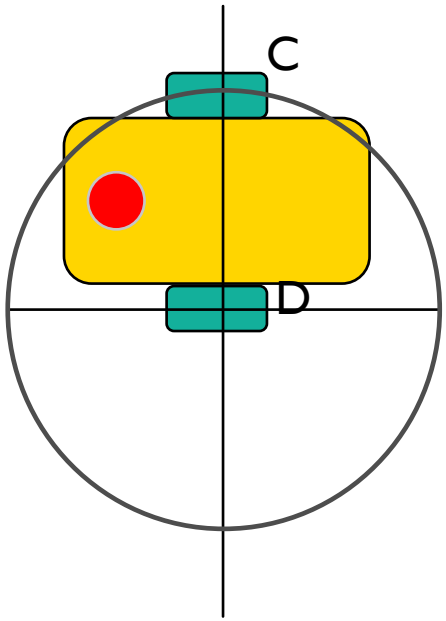
motor **rotations** for spinning the robot 360 degrees = (Cp /Cw)

motor **degrees** for spinning the robot 360 degrees = (Cp /Cw) * 360

In general terms, for turning "robot_turn" degrees:

motor_degrees = (Cp/Cw) * robot_turn

This works for **any** value of degrees. There is no special case for crossing 180, 360 or any other value, as we saw when using the yaw angle.



CHALLENGE - PIVOT

Write a function to make any pivot turn, clockwise or counterclockwise, using just MotorPair functions.

Use a parameter to make the speed variable

Write tests for your function

CHALLENGE – PIVOT: SOLUTION PAGE 1 OF 2

```
from hub import port
import motor_pair, runloop, sys, math

# Constants for Drive Base 1
motor_pair.pair(motor_pair.PAIR_1, port.C, port.D)
WHEEL_CIRCUMFERENCE = 17.5 # cm
TRACK = 11.2 #cm - please measure your own robot.
PIVOT_CIRCUMFERENCE = 2 * TRACK * math.pi

async def pivot_turn(robot_degrees, motor_speed):
    # Add a multiplier for gear ratios if you're using gears
    motor_degrees = int((PIVOT_CIRCUMFERENCE/WHEEL_CIRCUMFERENCE) * abs(robot_degrees))
    if robot_degrees > 0:
        # pivot clockwise
        await motor_pair.move_for_degrees(motor_pair.PAIR_1, motor_degrees, 50, velocity=motor_speed)
    else:
        #pivot counter clockwise
        await motor_pair.move_for_degrees(motor_pair.PAIR_1, motor_degrees, -50, velocity=motor_speed)
```

Define the imports, the globals that are fixed for your robot, and the pivot_turn function that computes the motor move degrees.

CHALLENGE – PIVOT: SOLUTION PAGE 2 OF 2

```
async def main():
    # Pivot 360 clockwise with speed 200
    await pivot_turn(360, 200)
    await runloop.sleep_ms(1000)
    # Pivot 360 counterclockwise with speed 200
    await pivot_turn(-360, 200)
    await runloop.sleep_ms(1000)
    sys.exit("Stopping")

runloop.run(main())
```

Write your tests in the main function. When you are satisfied that your code works, the function can be added to your library if you choose to use geometric rotation

TURNING WITHOUT GYRO – PROS AND CONS

Pros

Code is simple, and no special cases are needed when crossing 180, 360 etc. It works for any degrees in any direction

Cons

It depends on your robot wheel size, track size and gear ratio. Changing your robot means changing the code constants. The Yaw angle approach doesn't depend on robot geometry so much, though you will have to do some extra setup if your brick is oriented differently from DB1

This approach also works better at slower speeds and inaccuracies increase as speed increases.

The solution you choose depends on your situation. Run tests and find out what works best for you.

CREDITS

This lesson was created Sanjay and Arvind and Sanjay Seshan for Prime Lessons

Additional contributions by FLL Share & Learn community members

More lessons are available at www.primelessons.org