

WVPL 2022 Makers Club

Amazing Maze

Lessons 6-10

Olavi Kamppari
Feb 24, 2022

Lesson 6, Advanced Line Follower (SKIP OVER)

- The basic line follower can be implemented with a proportional control
 - $CV = \text{gain} * (PV - SP)$
- The accuracy can be improved in a case where a constant bias can be used
 - $CV = \text{gain} * (PV - SP) + \text{bias}$
- The bias is used as a special case of the generic iTerm (integral)
 - $CV = \text{pTerm} + \text{iTerm}$, where pTerm (proportional) is $\text{gain} * (PV - SP)$
- In a setup where special turns and change in speed is required
 - The first step is to determine what are the correct use cases
 - There are several ways to do that, such as
 - The CV is calculated based on gain, PV, and SP
 - When PV and SP are close, the basic proportional control is used
 - Otherwise special actions are required

Lesson 7, Line Follower with PD Control

- Instead of using a tailored Proportional Control with multiple use cases a generic Proportional Derivative Control can be used
- This is a special case of the generic PID control where
 - **CV = pTerm + iTerm + dTerm**
 - The pTerm is correcting the current deviation
 - The iTerm is correcting systematic past deviations
 - The dTerm is trying to prevent future deviations by dampening current changes
 - The deviation is the delta between PV and SP (**delta = PV - SP**)
- For consistency the tuning parameter name gain is replaced with the constant kP
 - **pTerm = kP * delta**
- The iTerm is a cumulative value of past delta values - not used in this case
 - $iTerm = iTerm + kI * delta$
- The dTerm is the change in the delta value
 - **dTerm = kD * (delta - prevDelta)**
 - **prevDelta = delta**

Lesson 7, Main Program

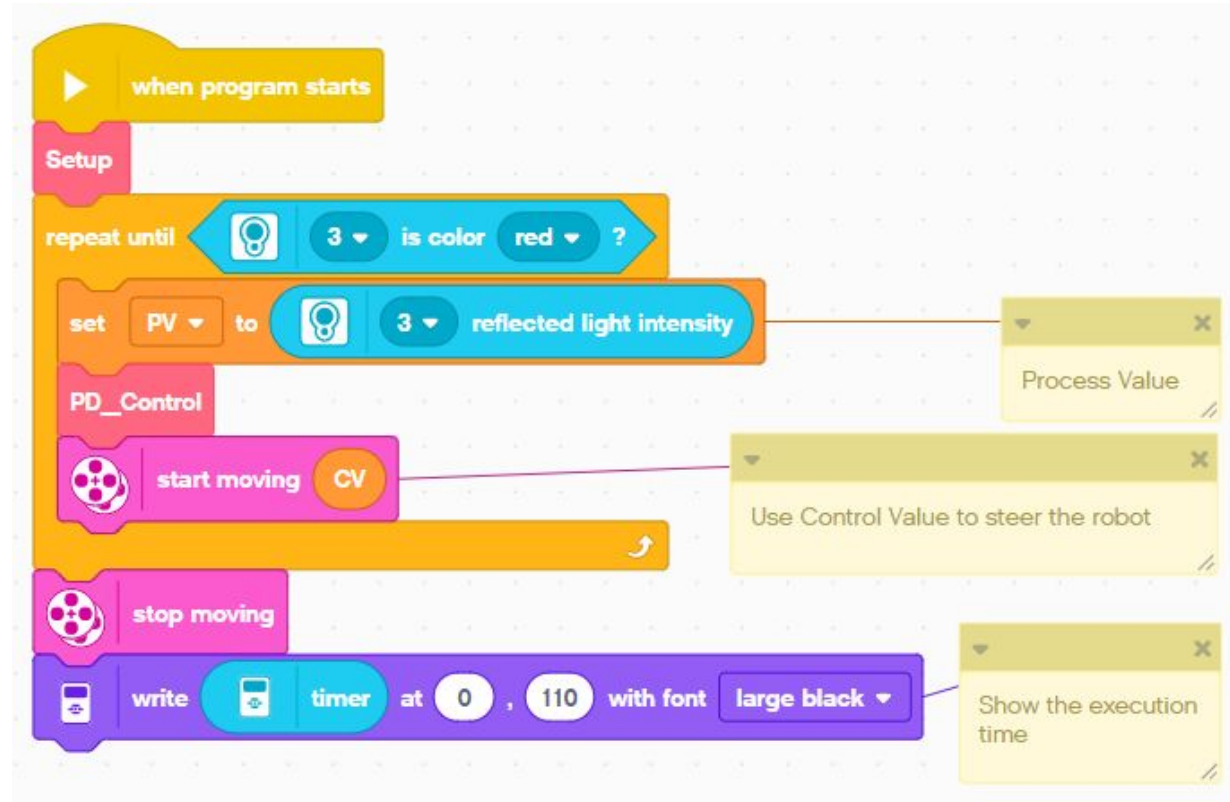
After starting, setup the variables.

Run the main loop, until the color sensor detects red.

Get the PV from the color sensor.

After calculating the CV in the PD_Control block use it to control the steering of the robot.

After main loop, stop and show time.



Lesson 7, Setup

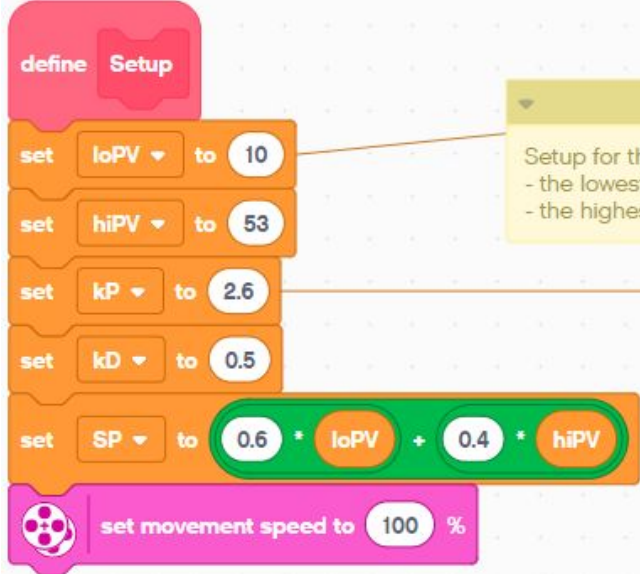
The Low and High PV values have to be measured with the robot to get the values like 10 and 53.

During the testing, adjust the kP value in range 1..5 and observe the impact.

Adjust the kD value in range 0..2 and observe the impact.

The SP should be towards the low value (the dark tape). The sum of the coefficients must be 1 (0.6+0.4).

Slow the speed below 100% if required.



The image shows a Scratch script for a robot's setup phase. The script is contained within a 'define Setup' block and consists of the following steps:

- set loPV to 10
- set hiPV to 53
- set kP to 2.6
- set kD to 0.5
- set SP to $0.6 * \text{loPV} + 0.4 * \text{hiPV}$
- set movement speed to 100 %

Two callout boxes provide additional context:

- Setup for the sensor range**
 - the lowest value (loPV) is for the black tape
 - the highest value (hiPV) is for the light background
- Init tuning parameters**
 - gain for proportional control (kP)
 - dampening for derivative control (kD)
 - Set Point between loPV and hiPV (target value for reflected light)
 - movement speed

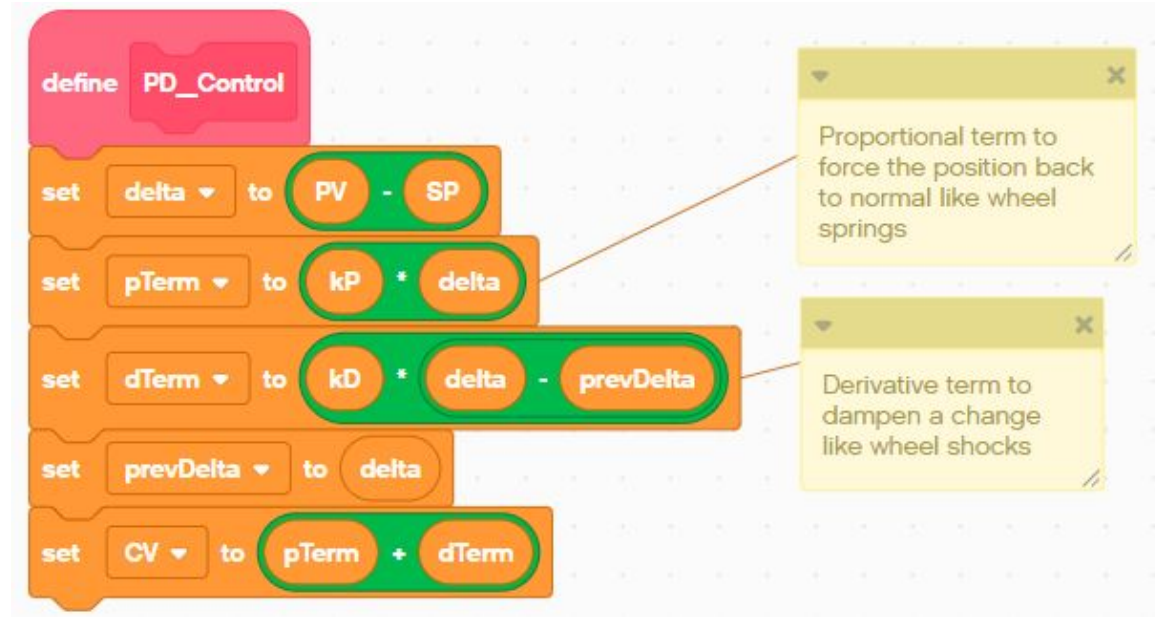
Lesson 7, PD Control

This is the PID controller without the I-Term. The obstacle course is too short and has too sharp turns to be able to use the I-Term.

The P-Term is trying to return the robot back to the edge of the black tape.

The D-Term is trying to slow down the turns when the sensor goes over the edge.

The prevDelta value is required to detect the changes in the delta value.



Lesson 7, Experimentation

Keep track of the sensitivity of the tuning parameters

Change movement speed, SP, kP, and kD by 5 % up and down

Record the % change in the time

For example by reducing the speed by 5 % the time could increase by 4 %

Change the sign of kP and kD values

and observe how the robot follows the right edge of the black tape.

Lesson 8, Wall Follower

- This case uses a simple proportional control where
 - The PV is the distance to the wall on the right
 - The CV is used for steering in the Start Moving block
- There is no bias, but special consideration is required for the long distance at the wall end.
 - The calculated CV value can be too large and that has to be prevented
 - An additional tuning parameter maxCV is used with the following logic
 - If $CV > \text{maxCV}$ then $CV = \text{maxCV}$
- The basic control is
 - $CV = kP * (PV - SP)$
- A good SP default value is 10
 - The robot is trying to stay 10 cm from the wall on right

Lesson 8, Main Program

After starting, setup the variables.

Run the main loop, until the color sensor detects white.

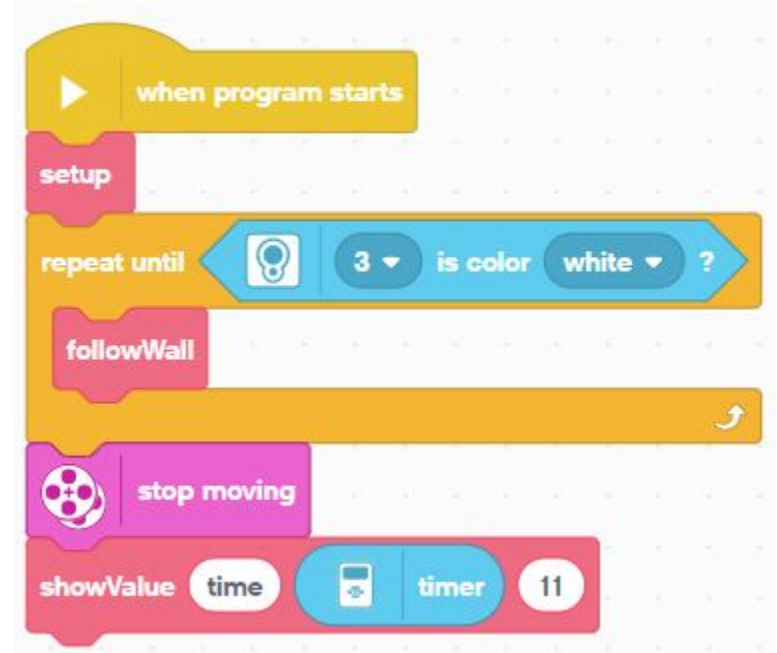
In the follow wall block

- Get the PV from distance sensor.

- Calculate the CV

- Use it to control the steering of the robot

After main loop, stop and show the time using show value block.



Lesson 8, Setup

Add variables for

kP (gain)

SP (setpoint = target distance to wall)

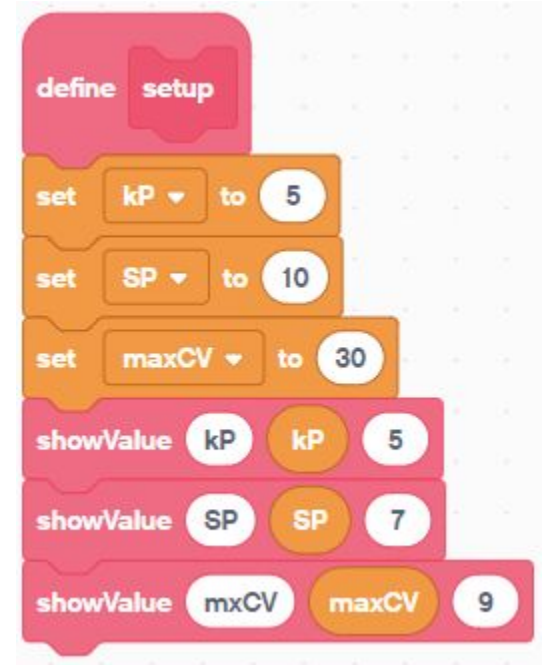
maxCV (maximum CV = steering to right)

Show the variables with labels on rows 5, 7, and 9

5: pixel rows 50 .. 69

7: pixel rows 70 .. 89

9: pixel rows 90 .. 109



Lesson 8, Follow Wall

Get the process value from distance sensor

The control value is the difference between PV and SP amplified with the proportional tuning value (kP)

The CV is limited to maxCV.

The PV could be as high as 200. With $kP = 5$ and $SP = 10$

The CV would be $5 \times (200 - 10) = 950$ leading to wall collision

Use CV as the steering value

Use show value blocks for PV and CV



Lesson 8, Show Value

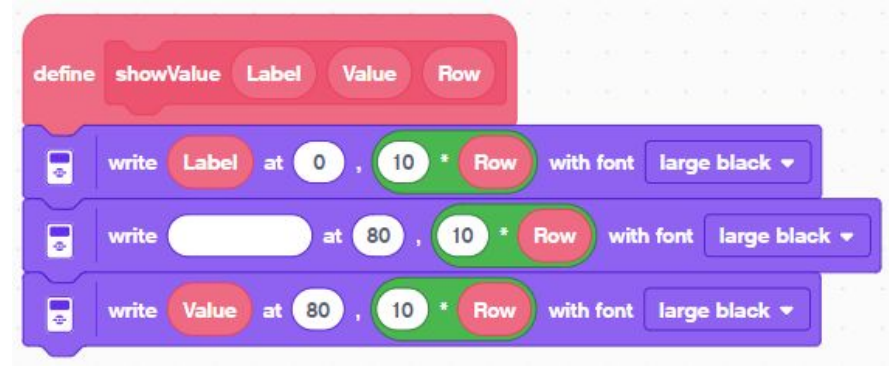
This routine shows the static tuning parameter values in the setup routine and the dynamic variables calculated in the Follow Wall routine.

The row number is converted into pixel row number with multiplication by 10.

The label is shown in pixel column 0.

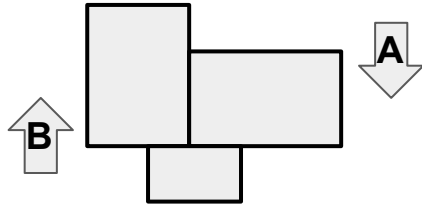
The value is shown in pixel column 80.

Before writing the value, the field is cleared with the white space.



Lesson 8, Around the block competition

- Arrange wooden crates on the floor with some degree of symmetry
- Robots A and B are competing at the same time
 - They do start at the same time on the opposing corners
 - The goal is to be faster than the other robot and after some rounds around the block to reach the other robot
- The winner will proceed to the semifinals and finals



Lesson 9, Wall Collision Handling

- The sensors available in the WVPL robot cannot avoid a wall collision
- A sensor is required to detect when a collision has happened
- A procedure is required to recover from the collision
 - If the wall following can avoid the collisions by always turning right
 - Then the right logic is to turn left after a collision
- The microsteps for this left turn are
 - Stop moving
 - Move backward to create space for the turn
 - Turn left
 - Continue to move forward and follow the wall on right
- The touch sensor on port 1 is used to detect the collisions

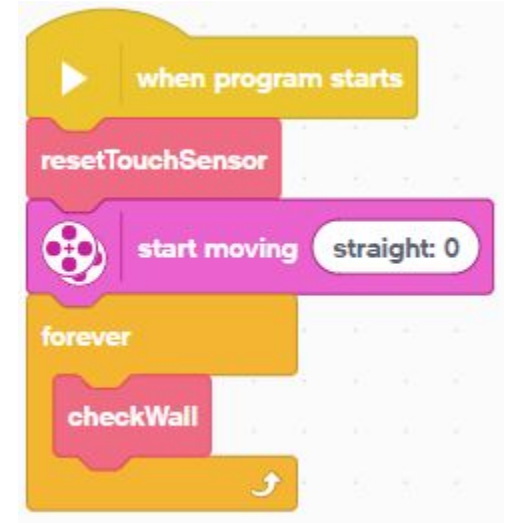
Lesson 9, Main Program

This is a different program than the earlier programs.

Once it is started, it is normally stopped by the red button in EV3 Classroom.

After starting, the touch sensor is reset and the robot starts to move forward.

In the infinite loop, the check wall block is continuously executed until the robot is stopped.



Lesson 9, Reset Touch Sensor

This is the same routine that was used in the loop moving lesson.



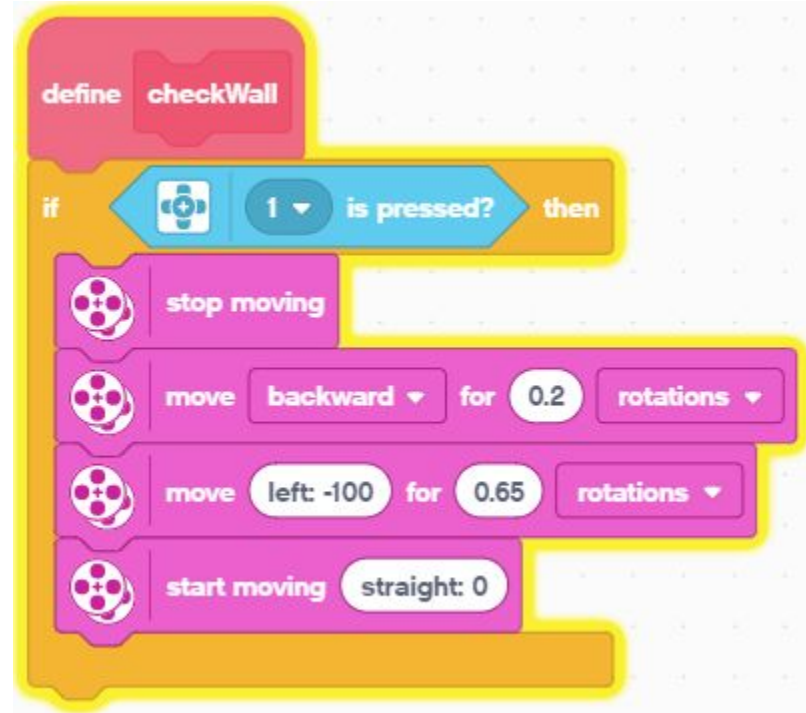
Lesson 9, Check Wall

The check wall actions are executed only if the touch sensor is triggered.

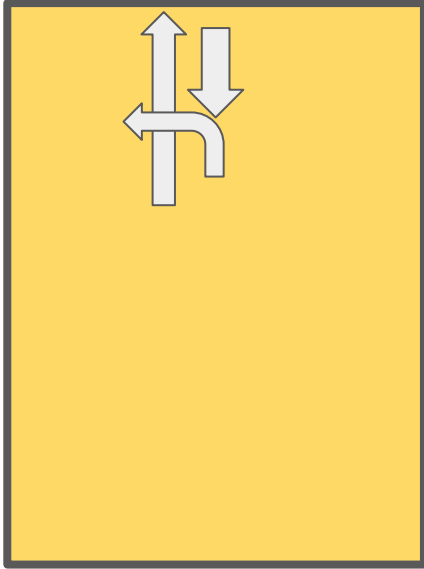
The first action is to stop.

By moving some centimeters backward there will be enough space to go forward around 10 cm and turn left.

After that the robot will continue to go forward.



Lesson 9, Check Wall testing



Put the robot in a box and start the program.

Observe, how the bumper wheels are forcing the robot perpendicular against the wall.

As soon as the touch sensor is triggered, the robot backs off and turns to right.

Adjust the movement values, so that the robot turns 360 degree after hitting 4 walls.

Modify the robot, if it can get stuck in a corner.

Check, how close to the right wall robot can start.

Lesson 10, Maze Resolver Design

- The basic idea is to combine the wall follower wall collision handling together
 - Keep the “finger” on the right wall and go forward
 - If the wall ends, turn right and follow the next wall
 - If the wall ends without possibility to turn right, then the robot will collide with wall in front
 - After wall collision
 - Back off, turn left, and follow the wall that was in front

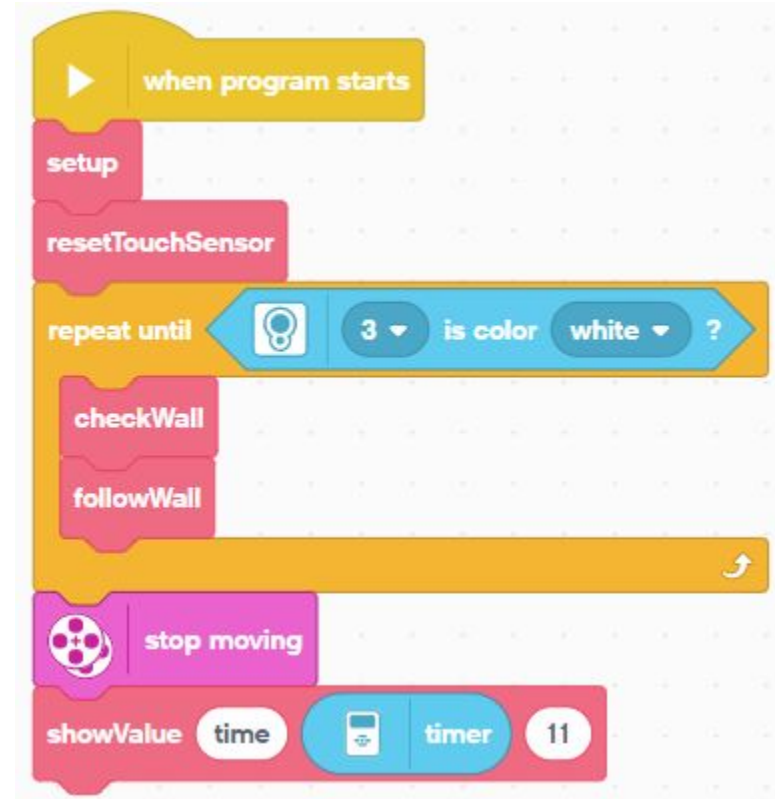
Lesson 10, Main Program

After starting, setup the control variables and reset the touch sensor.

Run the main loop, until the color sensor detects white.

In the main loop check the front wall always before following the wall on right.

After detecting white, stop moving and show the time on row 11.



Lesson 10, Setup

Add variables for

kP (gain)

SP (setpoint = target distance to wall)

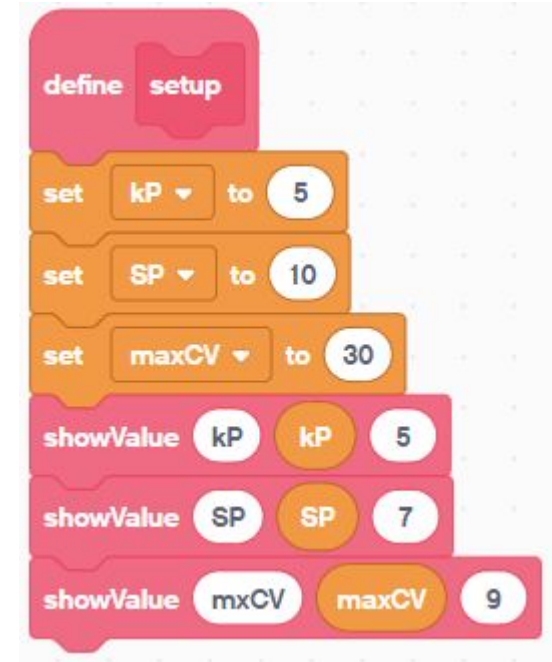
maxCV (maximum CV = steering to right)

Show the variables with labels on rows 5, 7, and 9

5: pixel rows 50 .. 69

7: pixel rows 70 .. 89

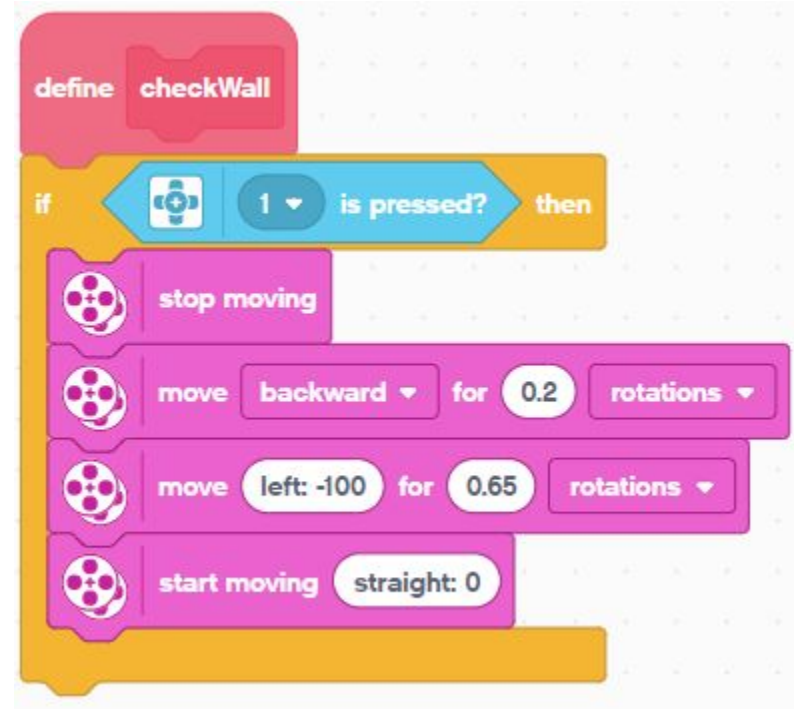
9: pixel rows 90 .. 109



Lesson 10, Check Wall

You should modify this routine to make it as fast as possible.

For example, you can try to back right a little bit and then turn left a little bit less.



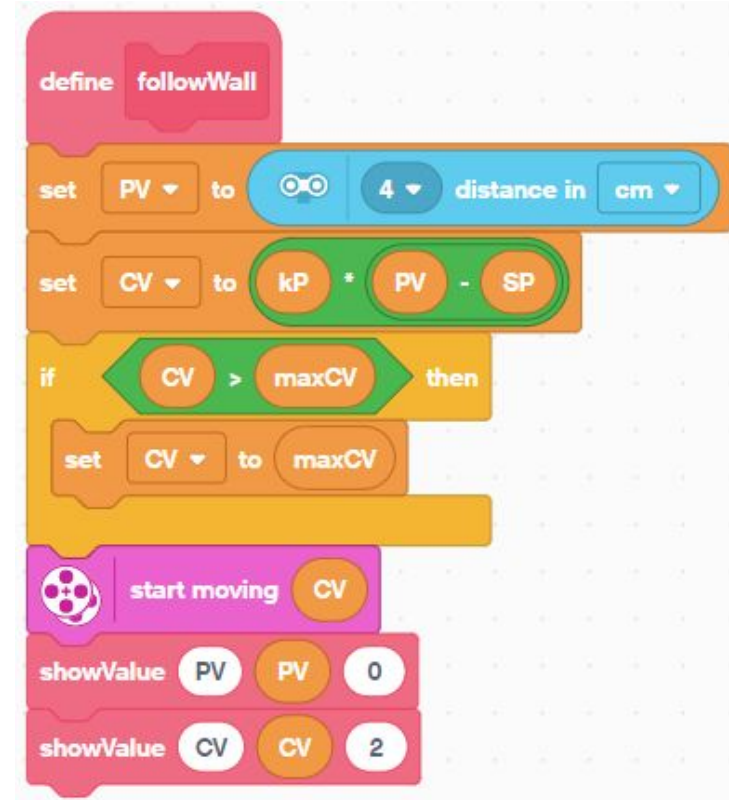
Lesson 10, Follow Wall

This is the main routine and should not be modified.

All required changes are done by changing the tuning values in the setup routine.

The process value (distance to wall) and control value (steering) are shown on display rows 0 and 2.

You can lift the robot and move it along the maze and use the steering value to simulate the robot actions.



Lesson 10, Collision Detection

For debugging purpose, create a sound when the touch sensor is triggered.

The touch sensor reset is the same as used in past lessons.



Lesson 10, Show Value

This routine shows the static tuning parameter values in the setup routine and the dynamic variables calculated in the Follow Wall routine.

The row number is converted into pixel row number with multiplication by 10.

The label is shown in pixel column 0.

The value is shown in pixel column 80.

Before writing the value, the field is cleared with the white space.

