UNIVERSITY · PRESENTATION 1 · LEARNING WITH PYTHON

Target by position

Meet the Advanced robot, then drive straight to a point with PositionX, PositionY, atan2d and angleDiff.

TheBotCup 2026 · AiBotCup Simulator · University



WHERE WE START

Sessions 1–11: the shared foundation

Presentations 0–5

Drive and sense

Wheels, Time, ultrasonic sensors, wall following.

Presentations 6–11

Mine and navigate

Collect, deposit, traps, the compass and angleDiff.

University 1 · today

Position

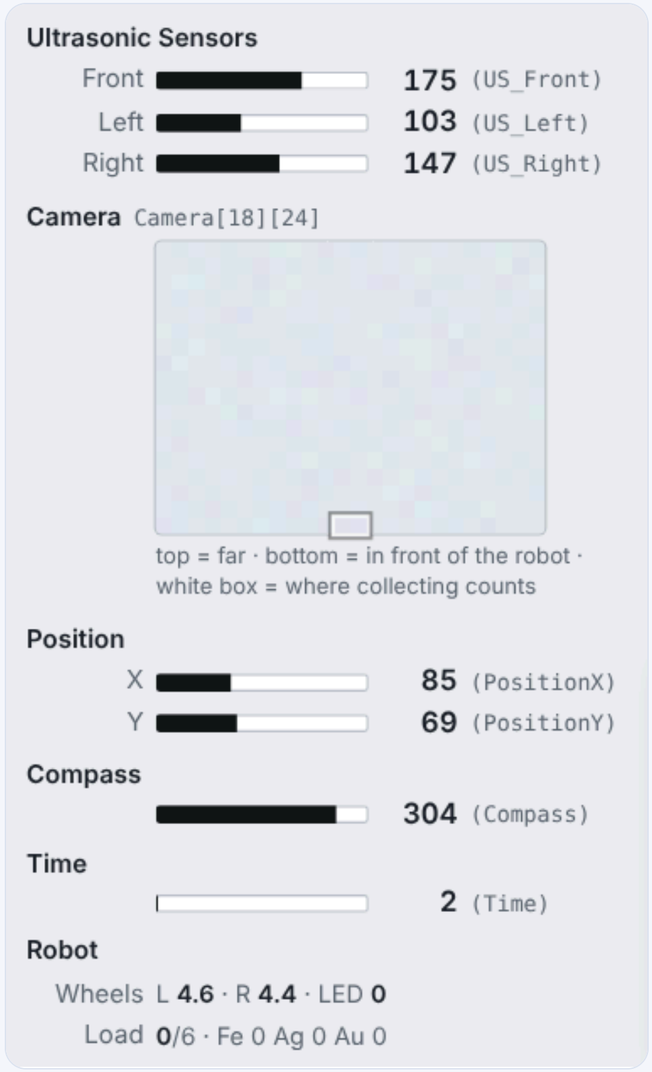
Know where you are, compute where to go.

Sessions 1–11 are the same for every category: use the school Python presentations P0–P11 for them.

PART 1

The Advanced robot

Camera instead of colour sensors · C or Python only



THE ADVANCED ROBOT

What it can sense

US_Front, US_Left, US_Right: distance to walls and blocks, cm.

Camera: an 18 × 24 image of the floor ahead. **No colour sensors.**

PositionX, PositionY: where the robot is, cm. **Compass**: 0–359°.

Time, **LoadedObjects** and more.

Programmed in **C or Python only**: no statements.

THE PROGRAM LOOP

One function, every 60 ms

```
count = 0                              # outside game1(): set once, kept

def game1():                           # runs every 60 ms
    WheelLeft = 3                      # -5.0 … 5.0
    WheelRight = 3
    Duration = 0                       # ms to hold these outputs
```

game1() runs every **60 ms**: read sensors, set outputs.

WheelLeft, WheelRight: −5.0 to 5.0. Also **LED_1**, **Duration**, **MyState**.

A variable written **outside** the function keeps its value between loops: the program’s memory.

Today we keep the target in two such variables: **TX** and **TY**.

PART 2

The mission: Session 12

Drive to the target at (185, 135)

Session 12 · Target by position

Use the position sensor to drive to the target at X = 185, Y = 135. The robot starts at (35, 35) facing north, so the target is not straight ahead: turn towards it first.

Reach the target: +200.

Next: target 1

Camera tip: Your robot has PositionX, PositionY and Compass like the others: aim with atan2d and angleDiff, then stop when you are within about 10 cm of the target. The pink pad also shows in the Camera image as you arrive.

Sessions | Restart session

SESSION 12 · TARGET BY POSITION

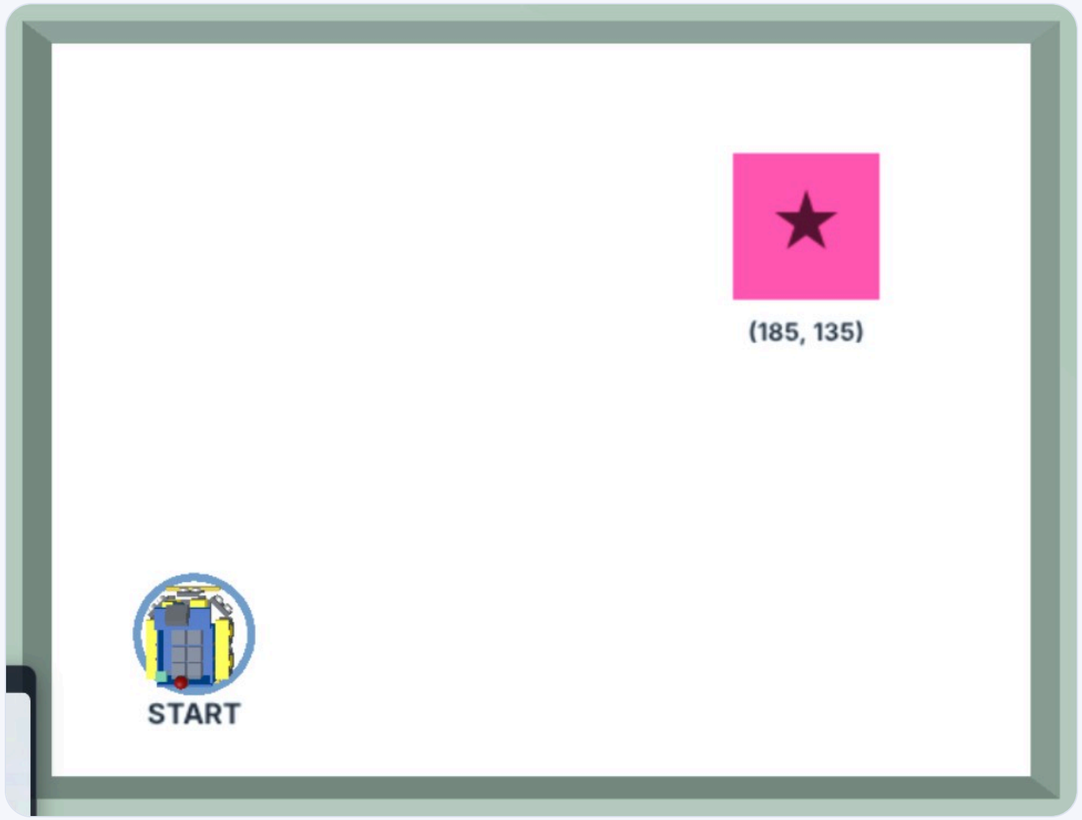
Turn towards it, then drive

Start at **(35, 35)**, facing **north**.

Target pad at **(185, 135)**: reach it for **+200**.

Time limit **3:00**.

Medals · Bronze: finish · Silver: under 1:48 · Gold: under 1:03 and perfect.



THE FIELD

An empty field and one pad

240 × 180 cm, nothing in the way.

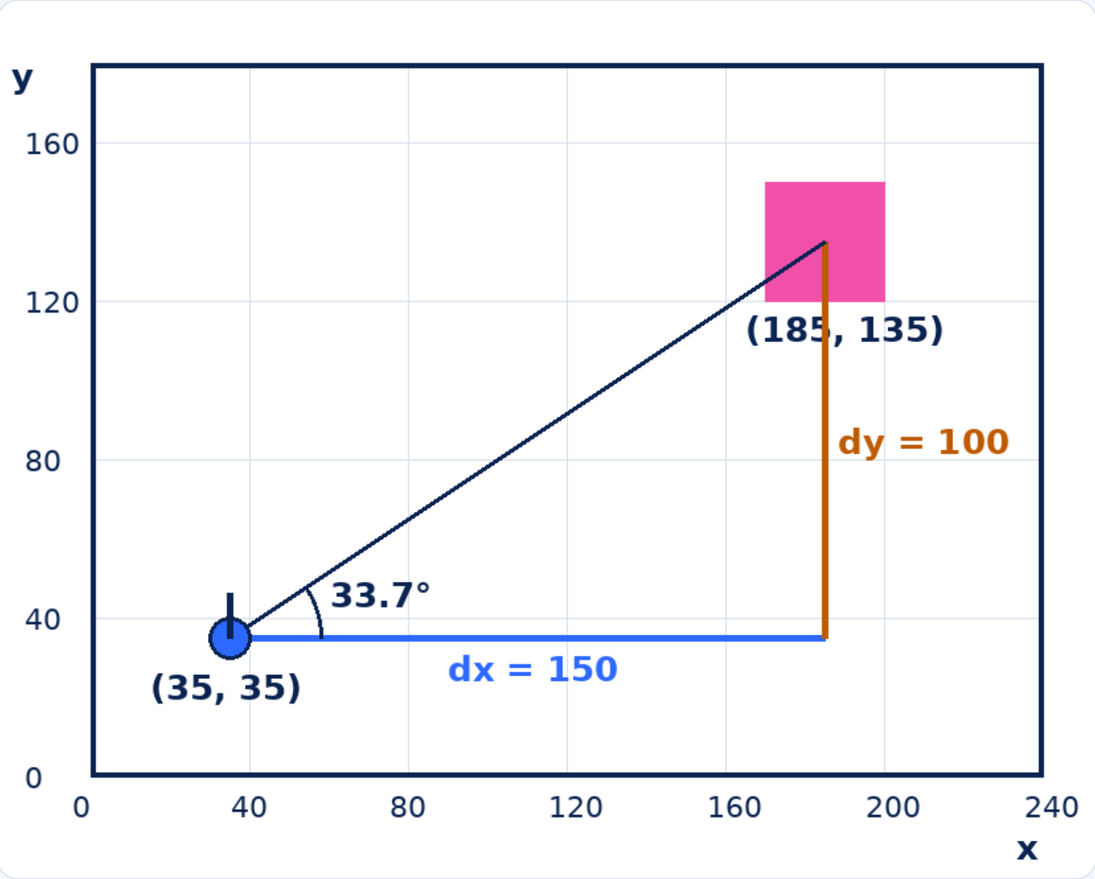
The pink pad is **30 × 30 cm**, centred on (185, 135).

No walls to follow, no colours to find: only **numbers**.

PART 3

The geometry

Coordinates, a heading and an error



GEOMETRY · COORDINATES

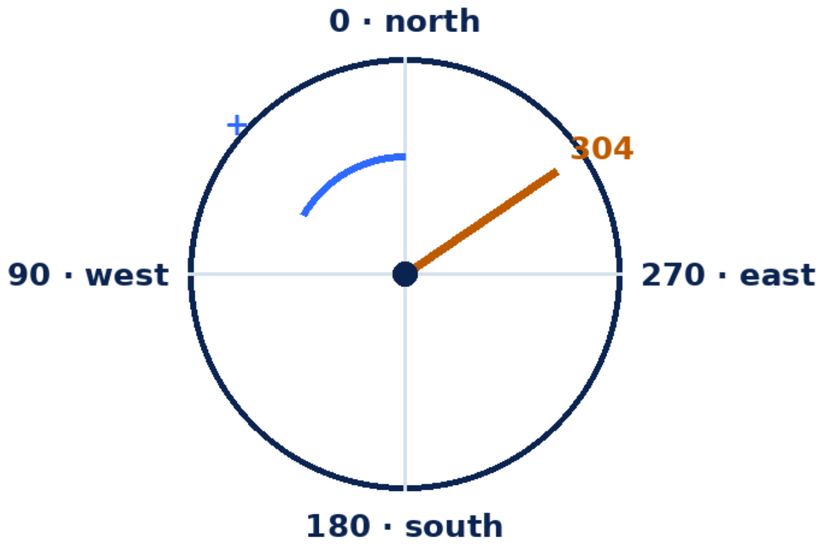
Where the robot is, where the target is

x grows to the right (east), **y** grows up (north), in cm.

dx = TX − PositionX = 185 − 35 = **150**

dy = TY − PositionY = 135 − 35 = **100**

Distance = $\sqrt{(dx^2 + dy^2)}$ = **180.3 cm**



GEOMETRY · THE HEADING

atan2d gives the angle; compass counts from north

atan2d(dy, dx) = atan2d(100, 150) = **33.7°**, measured from **east**, anticlockwise.

The **Compass** measures from **north**, anticlockwise: 0 N, 90 W, 180 S, 270 E.

So: heading = atan2d(dy, dx) − **90** = −56.3°, the same as **303.7°**.

In the tested run the robot drove at Compass **304**.

GEOMETRY · THE ERROR

angleDiff: how far, and which way, to turn

```
dx = 185 - 35                          # 150 cm to the right (east)
dy = 135 - 35                          # 100 cm up (north)
dist = sqrt(dx * dx + dy * dy)         # 180.3 cm
heading = atan2d(dy, dx) - 90          # 33.7 - 90 = -56.3, same as 303.7
err = angleDiff(heading, Compass)      # angleDiff(-56.3, 0) = -56.3
```

err > 8 → target on the **left**: turn left

err < −8 → on the **right**: turn right

−8 … 8 → **drive**, correcting gently

angleDiff(a, b) = a − b squeezed into −180 … 180. At the start: angleDiff(−56.3, 0) = **−56.3**: turn **right** 56°. No jump at 0/360: angleDiff(5, 355) = **10**.

PART 4

Step by step in Python

Five lines of maths, four branches

STEP 1 · THE PLAN

Every 60 ms: compute, then decide

1. Compute dx, dy, the distance, the heading and the error.
2. Closer than 6 cm → **stop**.
3. Error more than 8° → **turn on the spot** towards the target.
4. Otherwise → **drive** at 4.5, steering by 0.2 × error.

The robot recomputes everything every loop, so small drifts are corrected all the way.

STEP 2 · LINES 3–4

The target, kept in two variables

```
3  TX = 185                                    # target X (cm)
4  TY = 135                                    # target Y (cm)
```

Written **outside** game1(): set once, kept.

Change only these two numbers to drive somewhere else.

STEP 3 · LINES 6–11

Compute the distance and the error

```
 6  def game1():
 7      dx = TX - PositionX
 8      dy = TY - PositionY
 9      dist = sqrt(dx * dx + dy * dy)          # distance to the target
10      heading = atan2d(dy, dx) - 90           # compass heading to the target
11      err = angleDiff(heading, Compass)       # > 0: target is to the left
```

dx, dy: the two legs of the triangle.

heading: compass angle to the target.

err > 0: target on the left.

STEP 4 · LINES 12–19

Stop, turn, or drive and steer

```
12      if dist < 6:                                          # Arrived
13          WheelLeft = 0; WheelRight = 0
14      elif err > 8:                                         # TurnLeft
15          WheelLeft = -2; WheelRight = 2
16      elif err < -8:                                        # TurnRight
17          WheelLeft = 2; WheelRight = -2
18      else:                                                 # Drive, steering gently
19          WheelLeft = 4.5 - err * 0.2; WheelRight = 4.5 + err * 0.2
```

−2 / 2: turn on the spot.

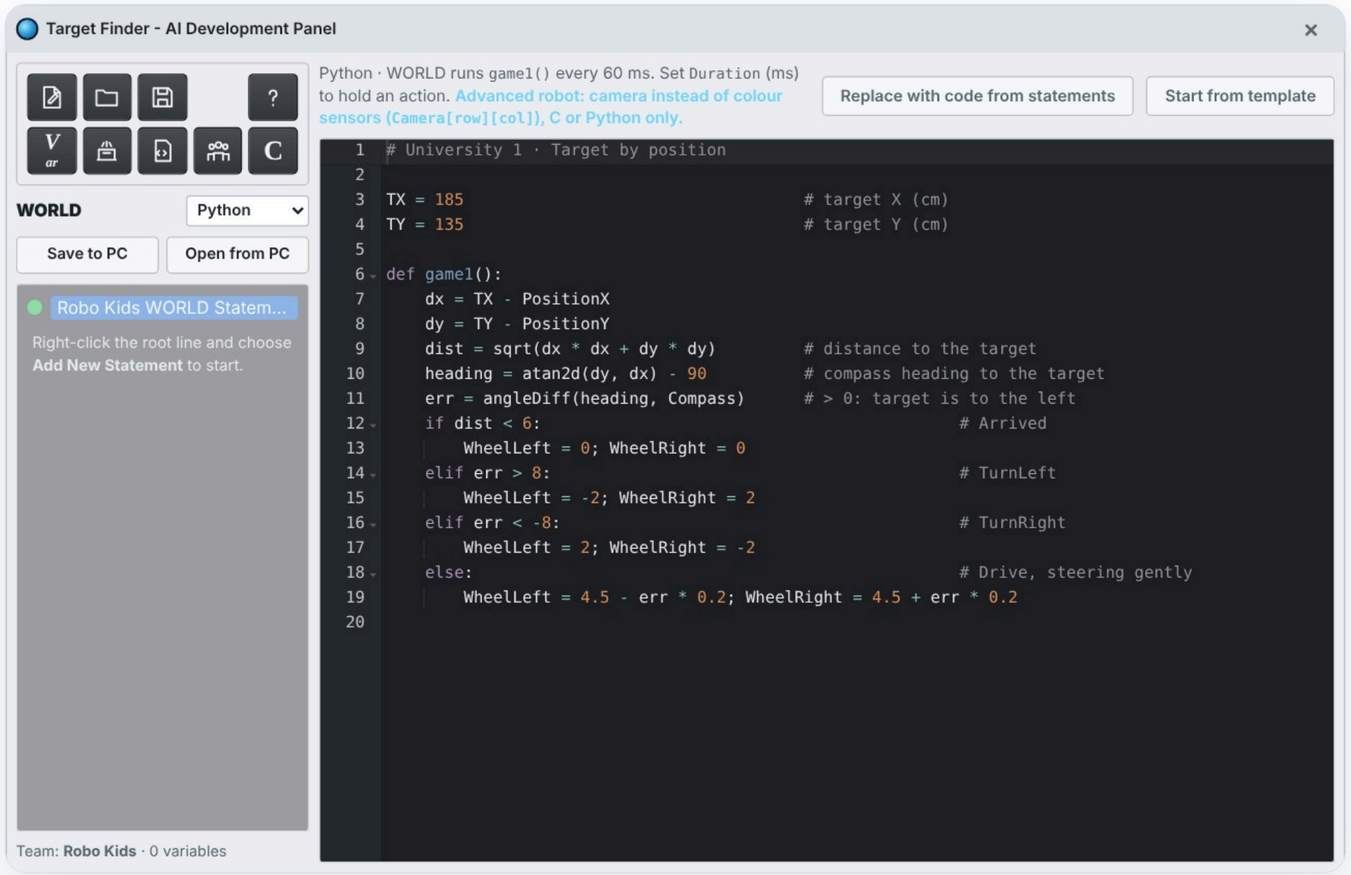
4.5 ∓ 0.2 × err: drive, nudging back on line.

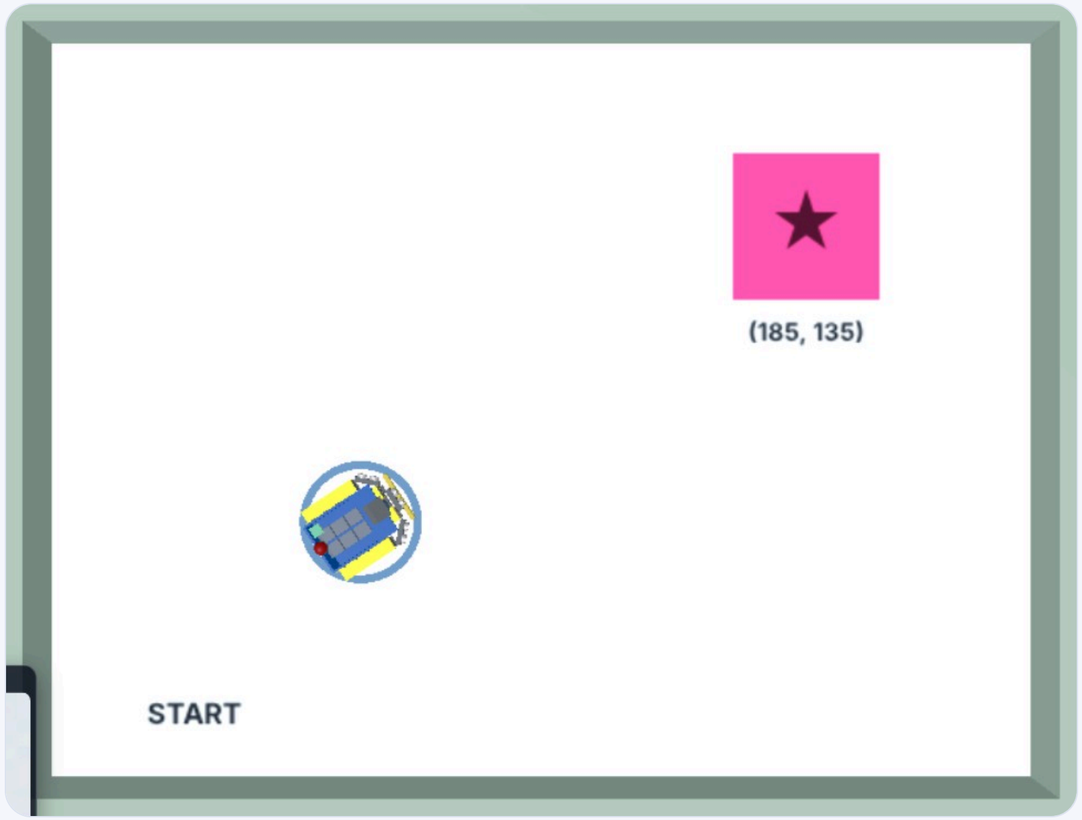
Positive err slows the left wheel: a gentle left turn.

STEP 5 · BUILD AND LOAD

The whole program, built

1. Category **Advanced**, Learning, Session 12.
2. WORLD menu: **Python**.
3. Type the program, name it **Target Finder**.
4. **Build**, then **Load** on BLUE.





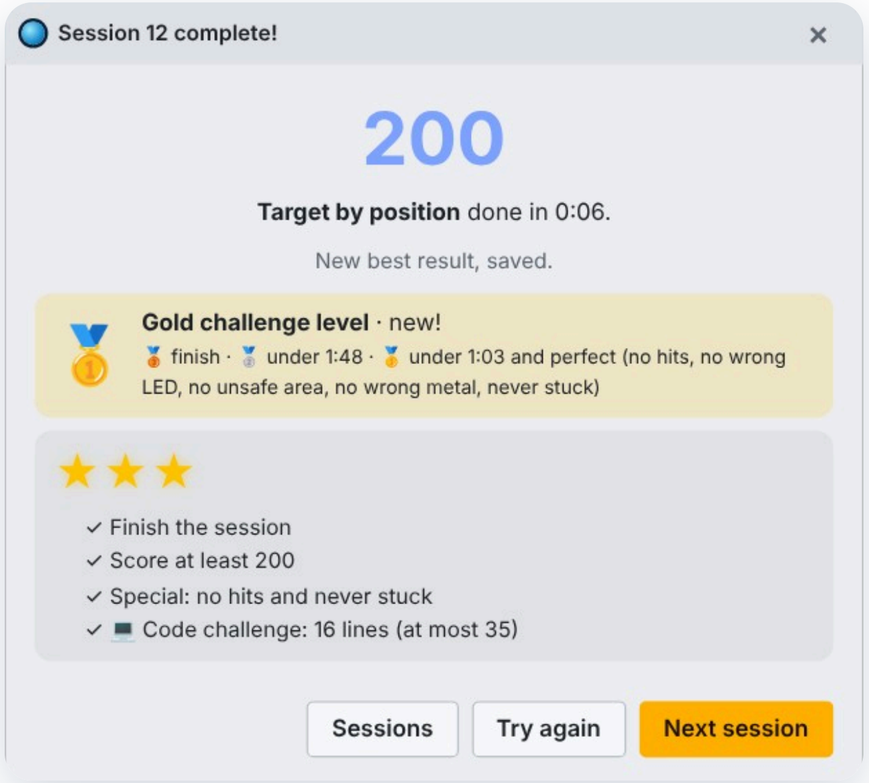
STEP 6 · RUN · 2.9 S

Aimed, and on its way

At **2.9 s** the robot is at **(88, 71)**, Compass **304**.

It first turned right on the spot, then drove straight at the pad.

The pad counts as soon as the robot drives onto it: **+200**.



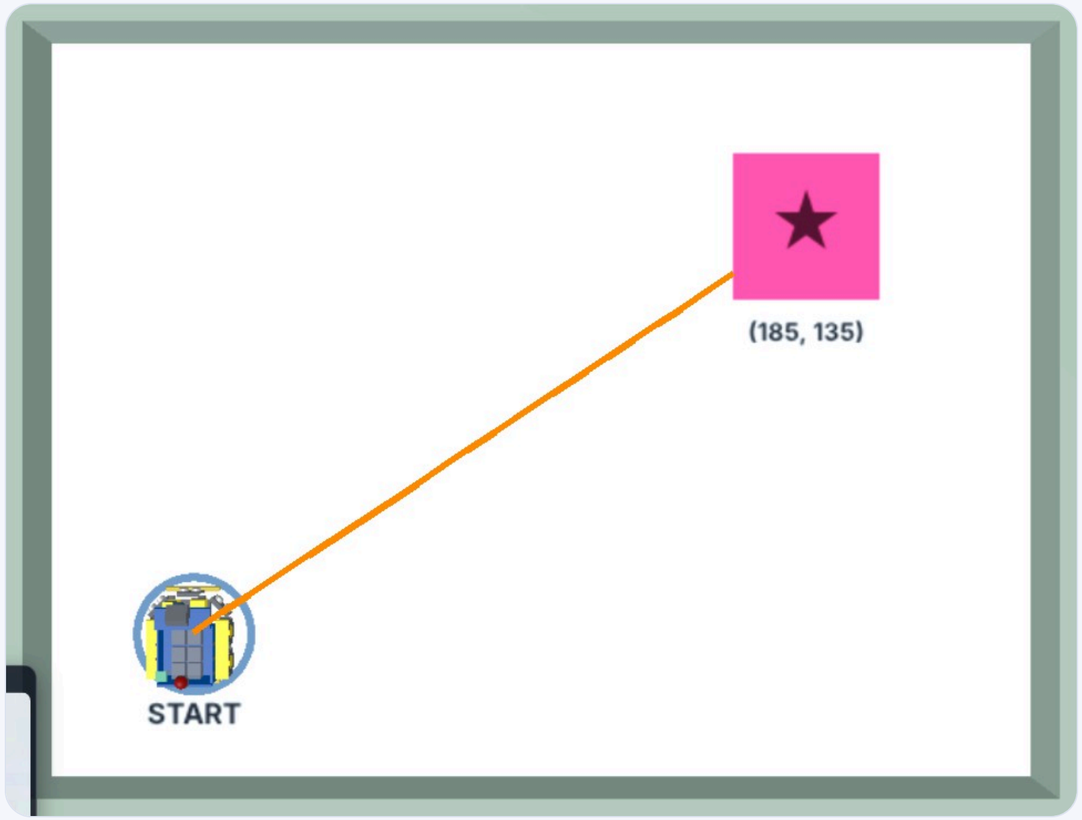
RESULT · TESTED

200 points in 0:06 · Gold

Medals · Silver: under 1:48 · Gold: under 1:03 and perfect.

Stars · finish · score 200 · no hits and never stuck.

Code challenge · at most 35 lines. We used 16.



THE PATH · RECORDED

A turn, then a straight line

No searching, no bouncing: the shortest way.

Compare: in Presentation 2 the turns were timed with the Time sensor. Here the robot **computes** them.

Position + geometry = **go anywhere**.

DID NOT WORK?

Check first, then tune carefully

What you see	What to do
Drives the wrong way	Check **atan2d(dy, dx)**: dy first! And the **− 90**
Turns the wrong way, spins forever	Swap the wheel signs: err > 8 means turn **left**
Zig-zags while driving	Lower the gain **0.2**, or widen the ±8° zone
Stops short of the pad	Make the stop distance smaller, or remove it
Build error with a line number	Check colons and indentation

NEXT TIME

University 2 · Two targets by position

Two pads in a fixed order: the program has to **change its own target** while it runs.