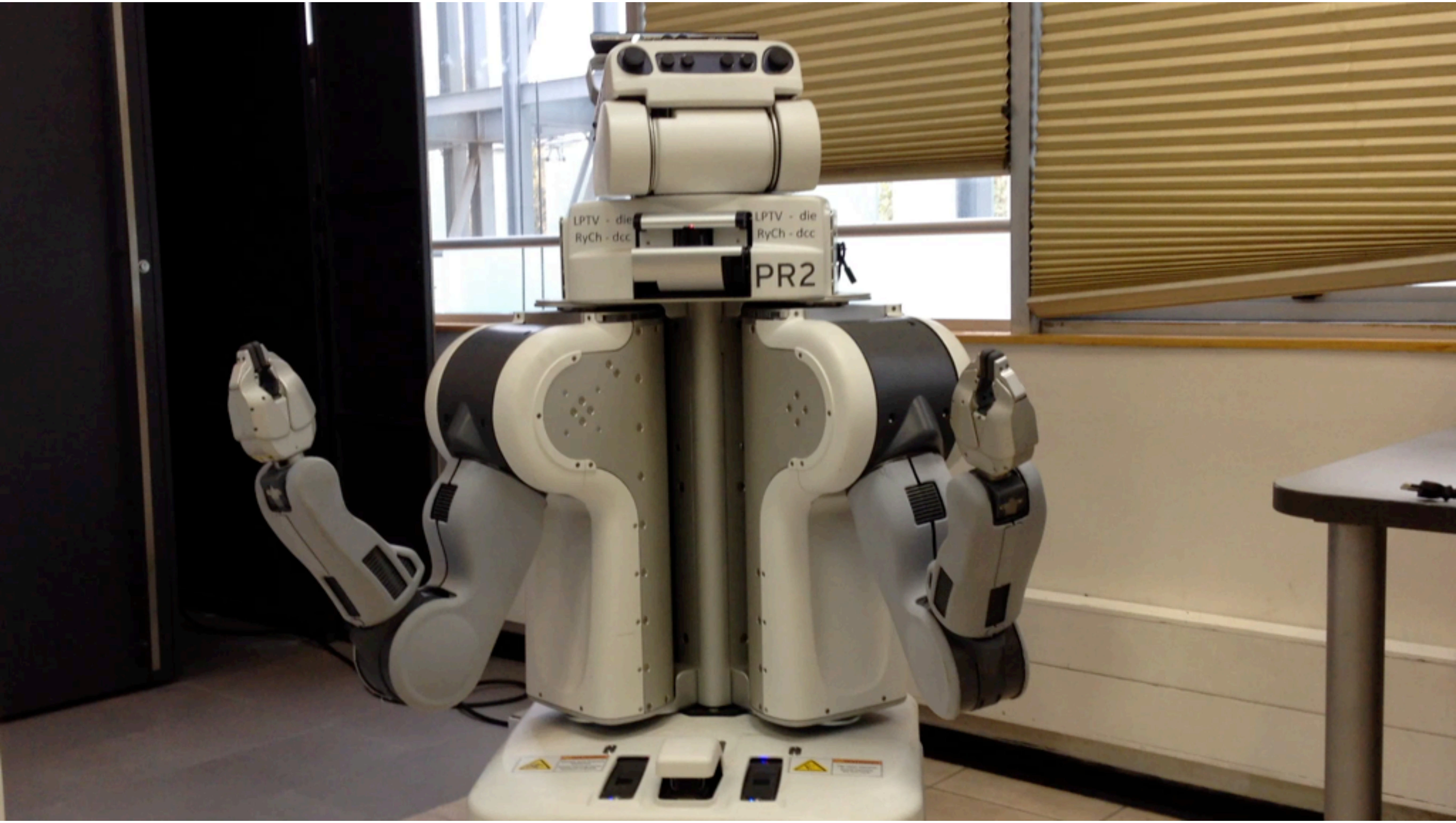
# Live Robot Programming

Johan Fabry,
Miguel Campusano, Pablo Estefó

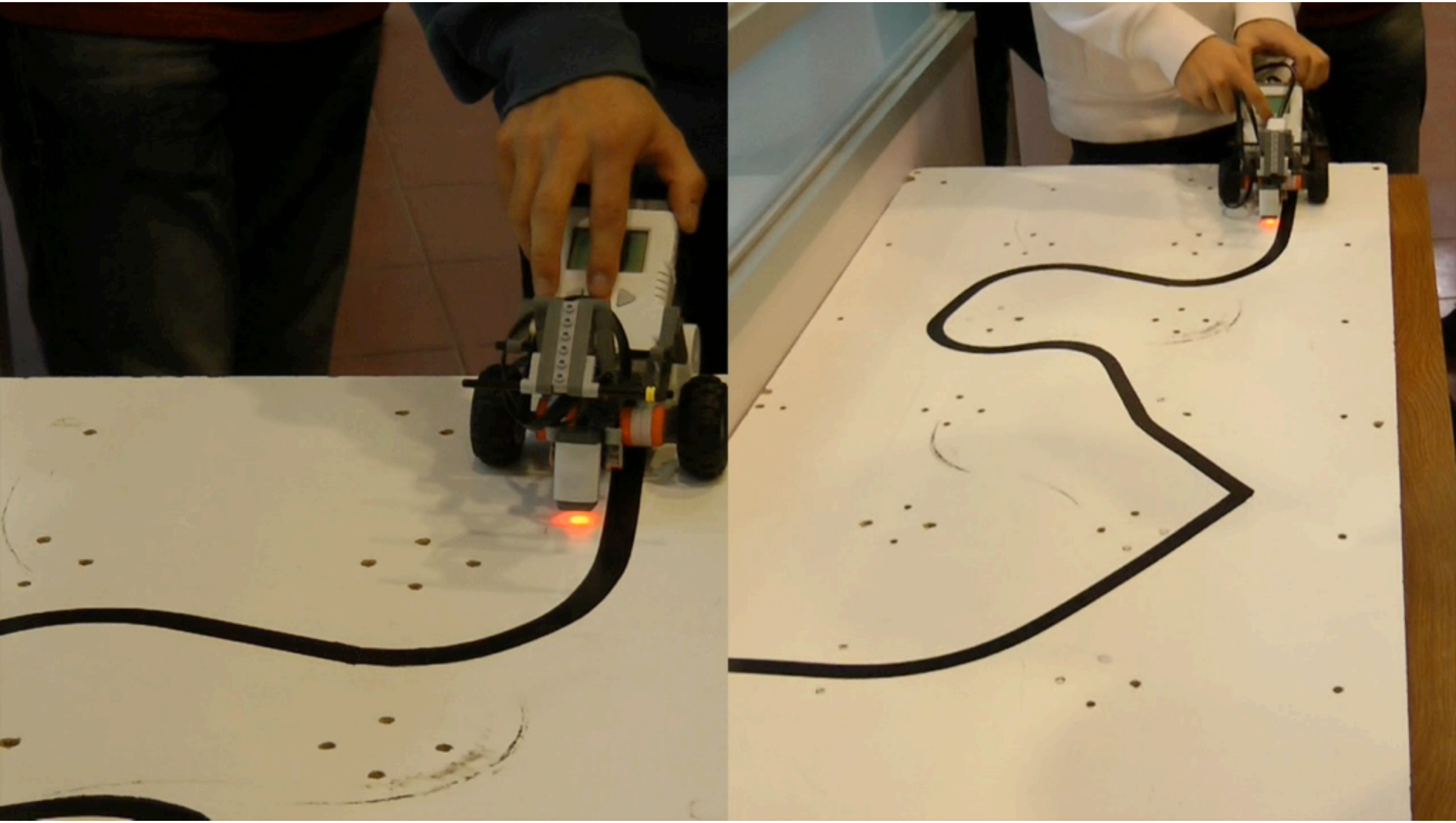
Pleiad & RyCh labs
Computer Science Department (DCC)
Universidad de Chile

# Two small stories

PR-J
PR2
Willow Garage
6X


LPTV - die
RyCh - dcc
LPTV - die
RyCh - dcc
PR2

Software is fundamental



Good Software
is fundamental

# Software Engineering

Time

=

Money

# Software Engineering

Time

=

Research

# Software Engineering

Time

=

Brainpower

# Software Engineering

Brainpower =
Problem complexity +
Technology complexity

# What do we want?

Waste less time in
**incidental** complexity

Use time on
**fundamental** complexity

# Example

“But **why** is the robot executing this behavior now?”

*(What is the internal state of the algorithm)*

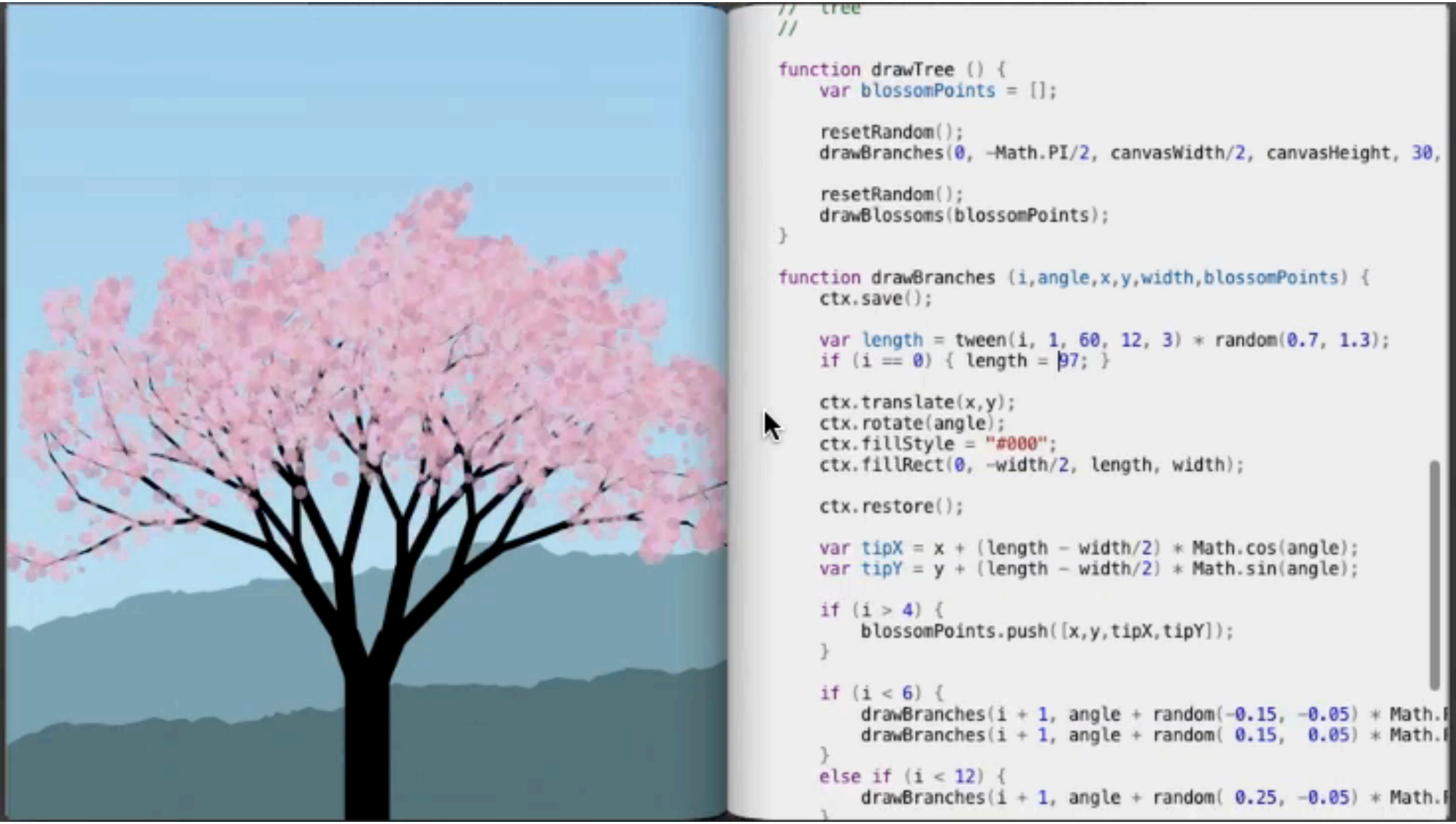
“**What would happen** if I change epsilon to 5 ?”

*(What are the correct parameters for the algorithm)*

Spend brainpower on the complexity of the task

Have an immediate connection to the behavior

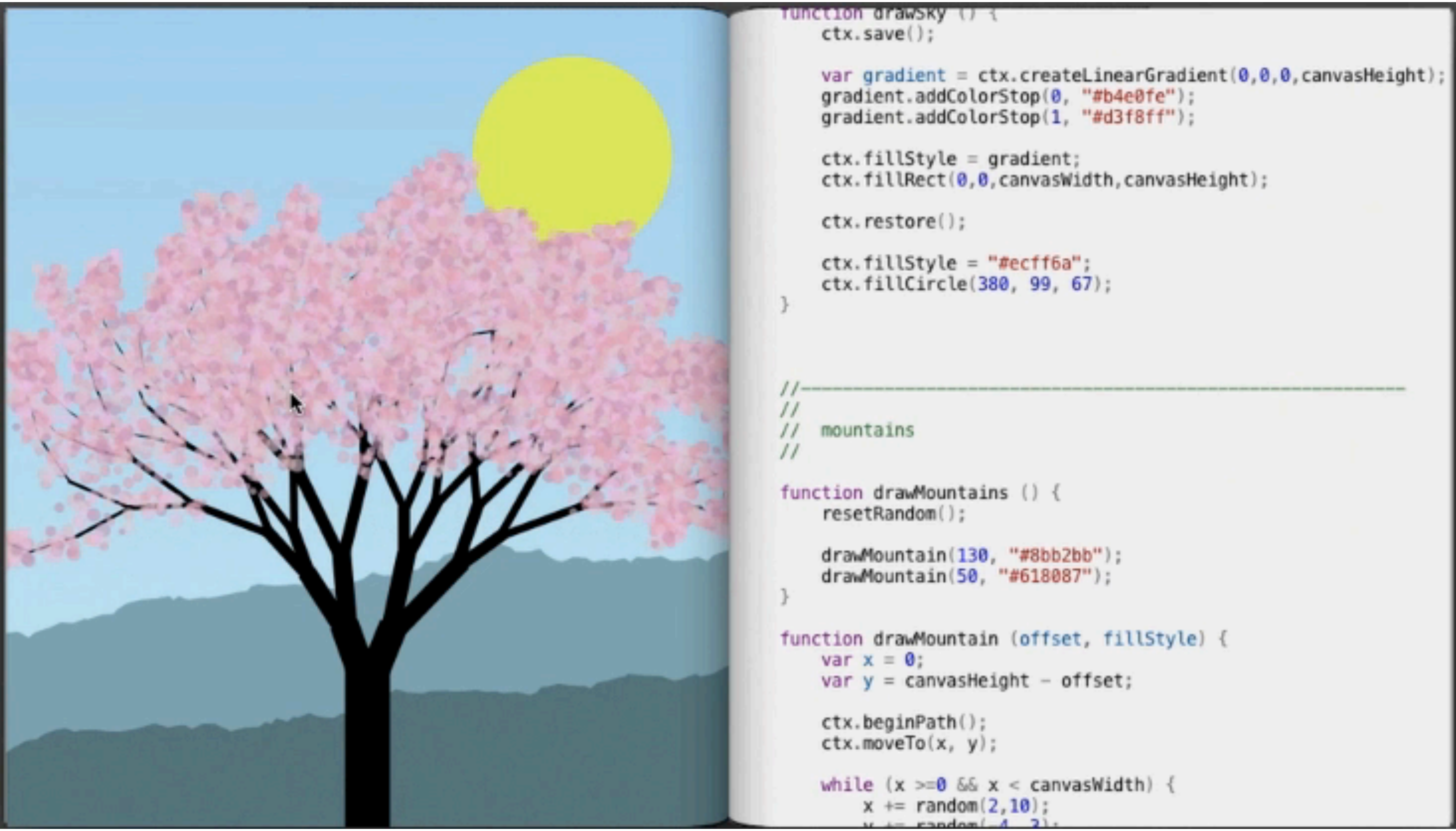
# Live Programming



```
//  tree
//

function drawTree () {
    var blossomPoints = [];

    resetRandom();
    drawBranches(0, -Math.PI/2, canvasWidth/2, canvasHeight, 30,

    resetRandom();
    drawBlossoms(blossomPoints);
}

function drawBranches (i,angle,x,y,width,blossomPoints) {
    ctx.save();

    var length = tween(i, 1, 60, 12, 3) * random(0.7, 1.3);
    if (i == 0) { length = 97; }

    ctx.translate(x,y);
    ctx.rotate(angle);
    ctx.fillStyle = "#000";
    ctx.fillRect(0, -width/2, length, width);

    ctx.restore();

    var tipX = x + (length - width/2) * Math.cos(angle);
    var tipY = y + (length - width/2) * Math.sin(angle);

    if (i > 4) {
        blossomPoints.push([x,y,tipX,tipY]);
    }

    if (i < 6) {
        drawBranches(i + 1, angle + random(-0.15, -0.05) * Math.P
        drawBranches(i + 1, angle + random( 0.15,  0.05) * Math.P
    }
    else if (i < 12) {
        drawBranches(i + 1, angle + random( 0.25, -0.05) * Math.P
    }
```

Bret Victor - Inventing on Principle (CUSEC 2012)

# Immediate Connection



```
function drawSky () {
    ctx.save();

    var gradient = ctx.createLinearGradient(0,0,0,canvasHeight);
    gradient.addColorStop(0, "#b4e0fe");
    gradient.addColorStop(1, "#d3f8ff");

    ctx.fillStyle = gradient;
    ctx.fillRect(0,0,canvasWidth,canvasHeight);

    ctx.restore();

    ctx.fillStyle = "#ecff6a";
    ctx.fillCircle(380, 99, 67);
}



//-----------------------------------------------------------------
//
//  mountains
//

function drawMountains () {
    resetRandom();

    drawMountain(130, "#8bb2bb");
    drawMountain(50, "#618087");
}

function drawMountain (offset, fillStyle) {
    var x = 0;
    var y = canvasHeight - offset;

    ctx.beginPath();
    ctx.moveTo(x, y);

    while (x >=0 && x < canvasWidth) {
        x += random(2,10);
```

Bret Victor - Inventing on Principle (CUSEC 2012)

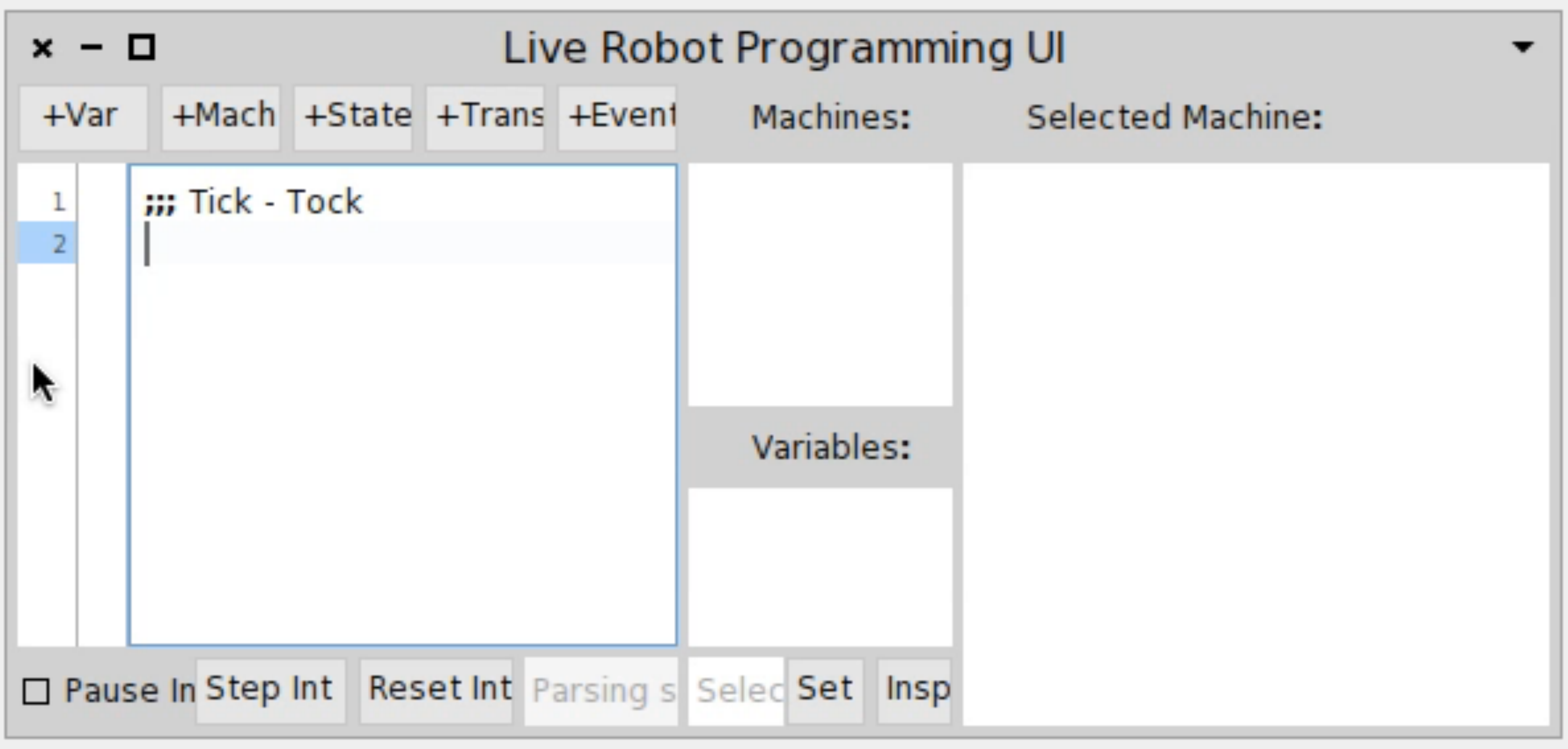
# Immediate Connection

# Live Robot Programming

# Fundamentals

- Live Programming Language
- For the behavior layer of robots
- Nested State Machines

# Machines, States



```
;;; Tick - Tock
```

# Immediate Connection

# Variables, Actions

Demo time!

# Immediate Connection

More about the language ~~in the paper~~ on the website.

http://pleiad.cl/LRP

# Conclusions

- Live Robot Programming: Yes you can!
- State machines are resilient
- Experience: radical speedup

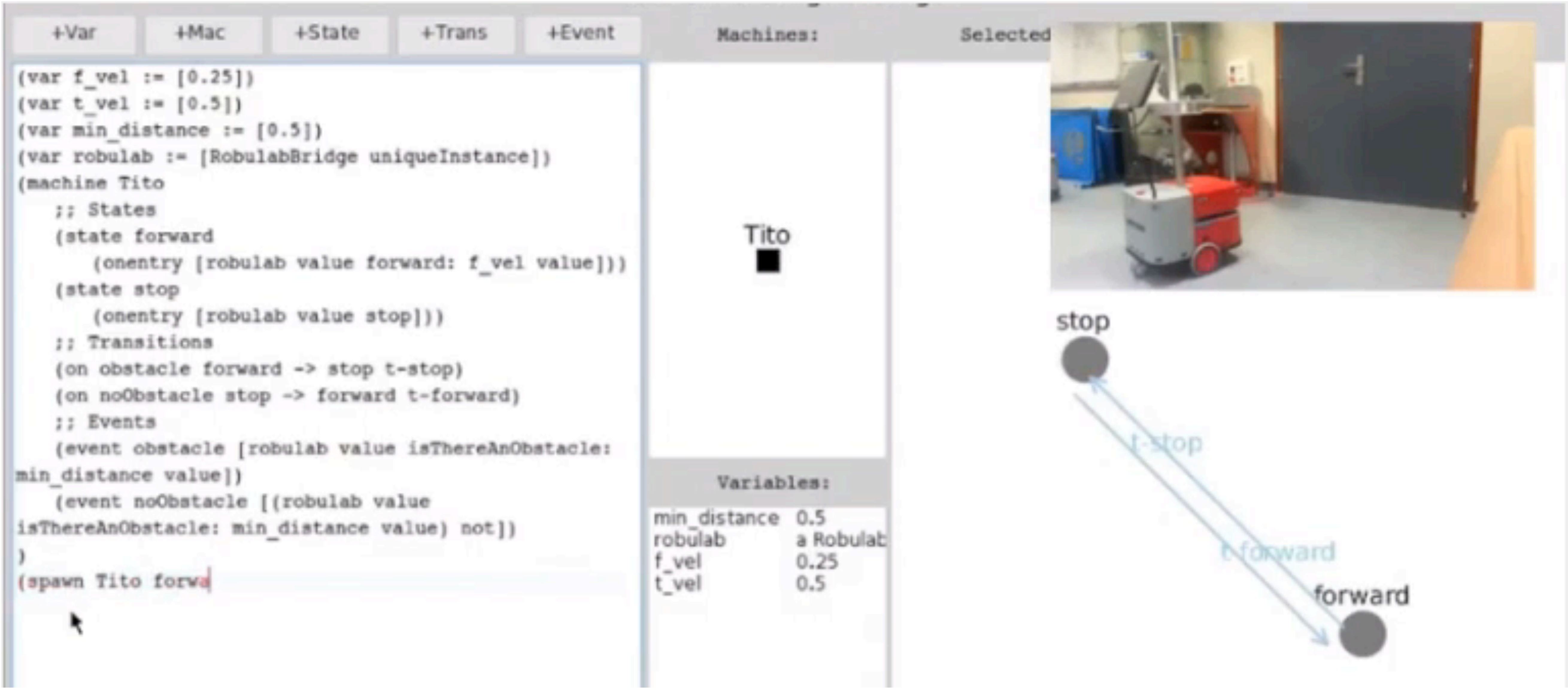
# Immediate Connection

# Future Work

- Refactorings to avoid restarts
- Test expressibility of the language
- Modularity and reuse of behaviors

http://pleiad.cl/LRP

# Active State

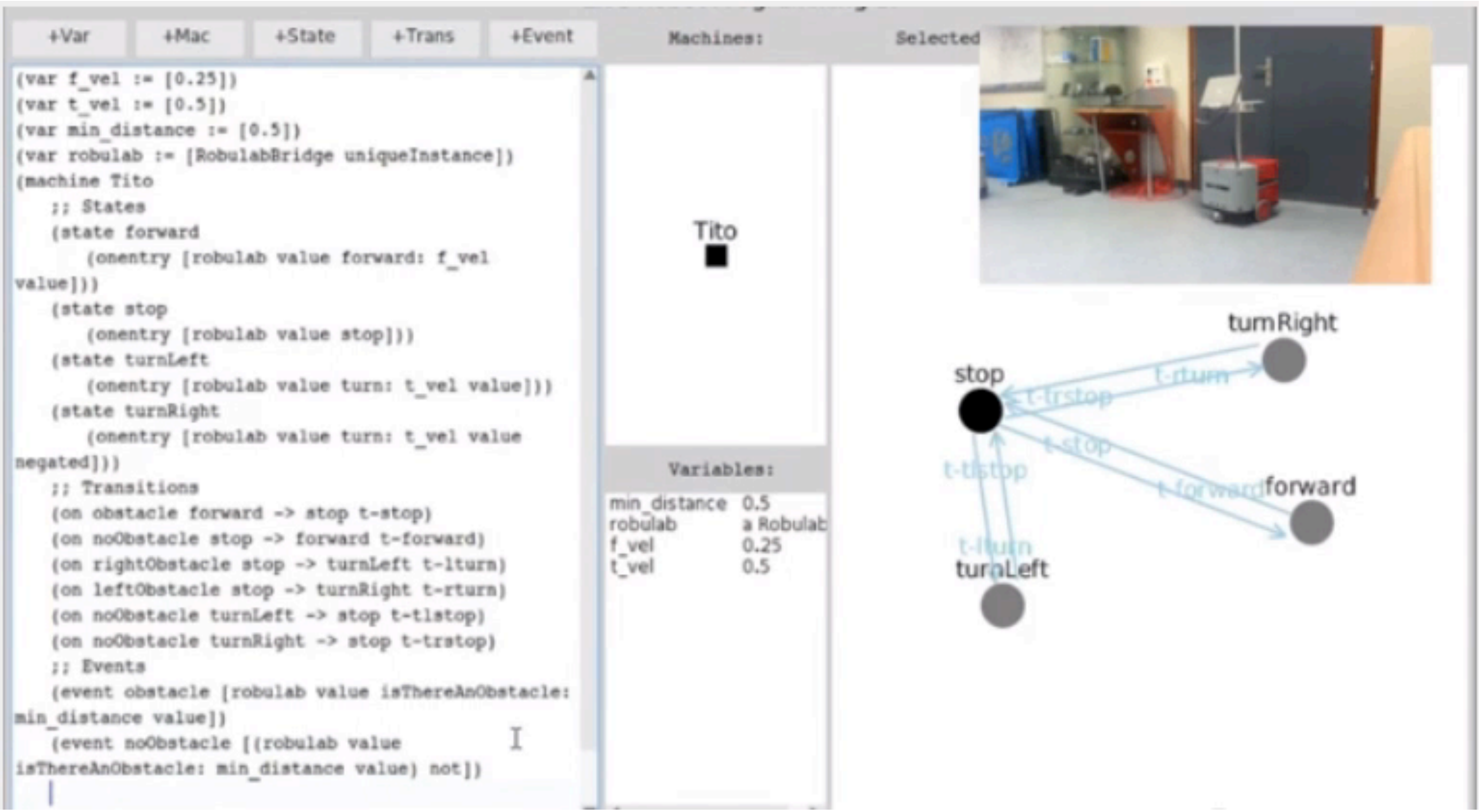


```
(var f_vel := [0.25])
(var t_vel := [0.5])
(var min_distance := [0.5])
(var robulab := [RobulabBridge uniqueInstance])
(machine Tito
    ;; States
    (state forward
        (onentry [robulab value forward: f_vel value]))
    (state stop
        (onentry [robulab value stop]))
    ;; Transitions
    (on obstacle forward -> stop t-stop)
    (on noObstacle stop -> forward t-forward)
    ;; Events
    (event obstacle [robulab value isThereAnObstacle:
min_distance value])
    (event noObstacle [(robulab value
isThereAnObstacle: min_distance value) not])
)
(spawn Tito forwa
```

Variables:

| | |
|---|---|
| min_distance | 0.5 |
| robulab | a Robulab |
| f_vel | 0.25 |
| t_vel | 0.5 |

# Active State



```
(var f_vel := [0.25])
(var t_vel := [0.5])
(var min_distance := [0.5])
(var robulab := [RobulabBridge uniqueInstance])
(machine Tito
   ;; States
   (state forward
      (onentry [robulab value forward: f_vel
value]))
   (state stop
      (onentry [robulab value stop]))
   (state turnLeft
      (onentry [robulab value turn: t_vel value]))
   (state turnRight
      (onentry [robulab value turn: t_vel value
negated]))
   ;; Transitions
   (on obstacle forward -> stop t-stop)
   (on noObstacle stop -> forward t-forward)
   (on rightObstacle stop -> turnLeft t-lturn)
   (on leftObstacle stop -> turnRight t-rturn)
   (on noObstacle turnLeft -> stop t-tlstop)
   (on noObstacle turnRight -> stop t-trstop)
   ;; Events
   (event obstacle [robulab value isThereAnObstacle:
min_distance value])
   (event noObstacle [(robulab value
isThereAnObstacle: min_distance value) not])
```

Variables:

| | |
|---|---|
| min_distance | 0.5 |
| robulab | a Robulab |
| f_vel | 0.25 |
| t_vel | 0.5 |

# Immediate Connection