

# Robótica Computacional

Comportamento, Controle, Localização

# Comportamento

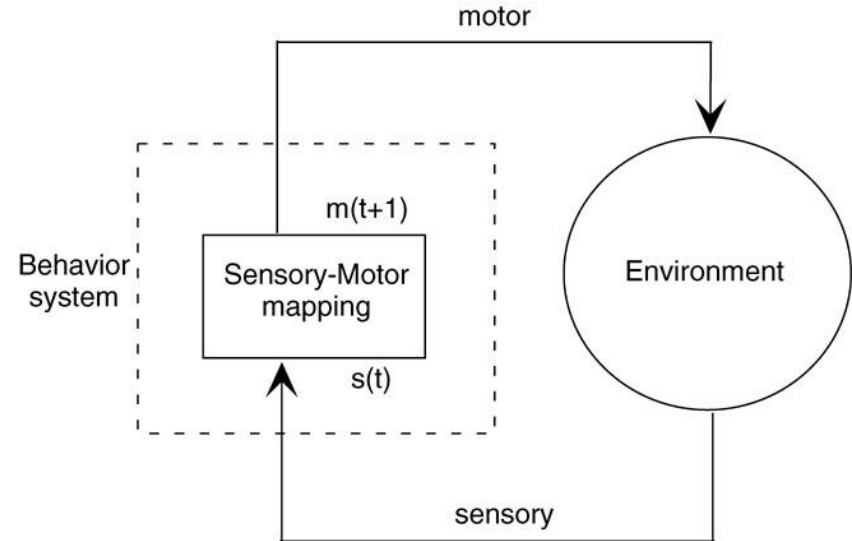
# Como decidir o comportamento do robô?

Na primeira aula discutimos o seguinte cenário. Qual ação o robô deveria tomar?

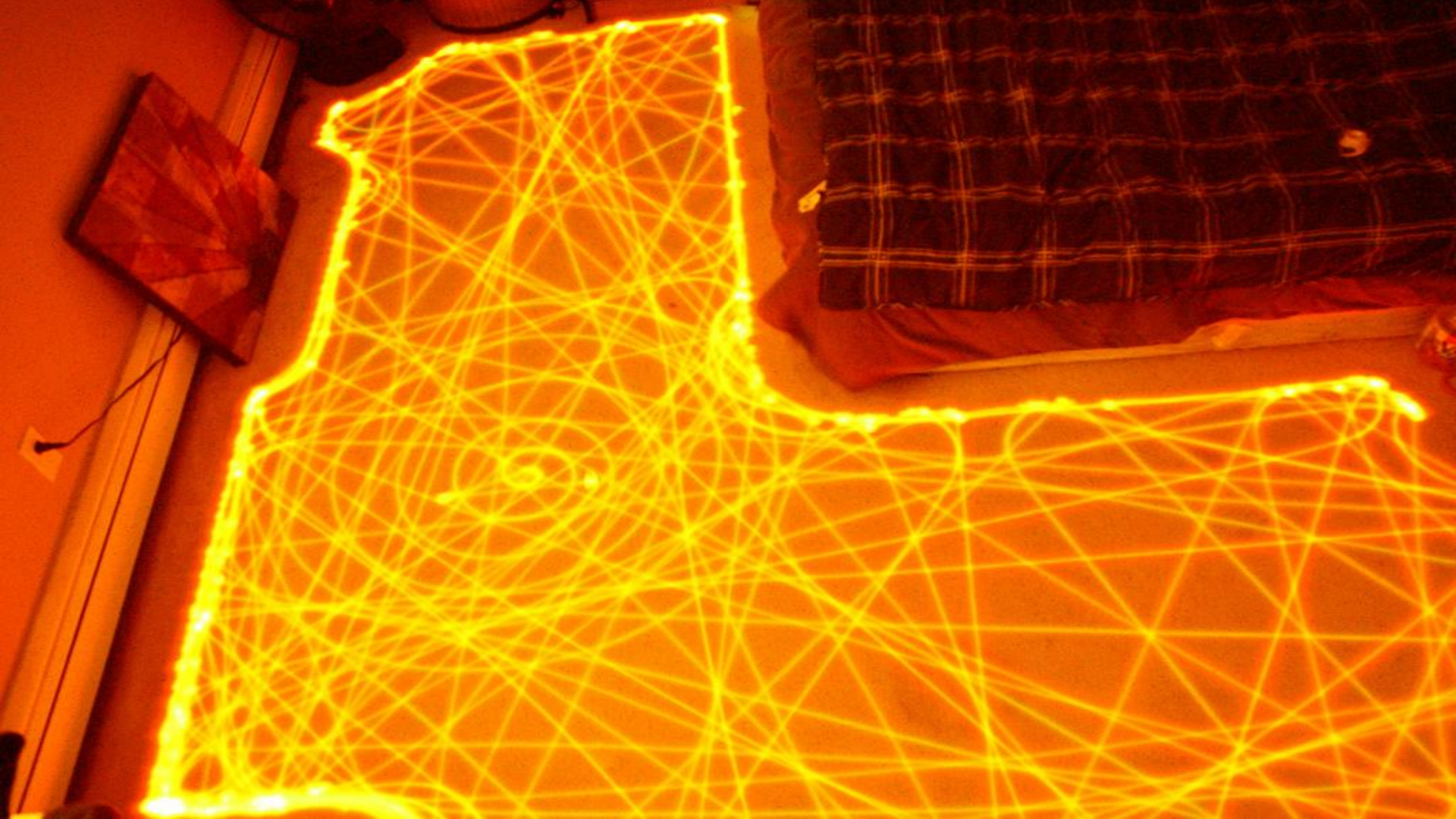


# Estrutura de controle de robô

- Um robô autônomo vê o ambiente ao redor e depois executa uma ação.
- Dependendo do seu comportamento, podemos ter diferentes resultados.



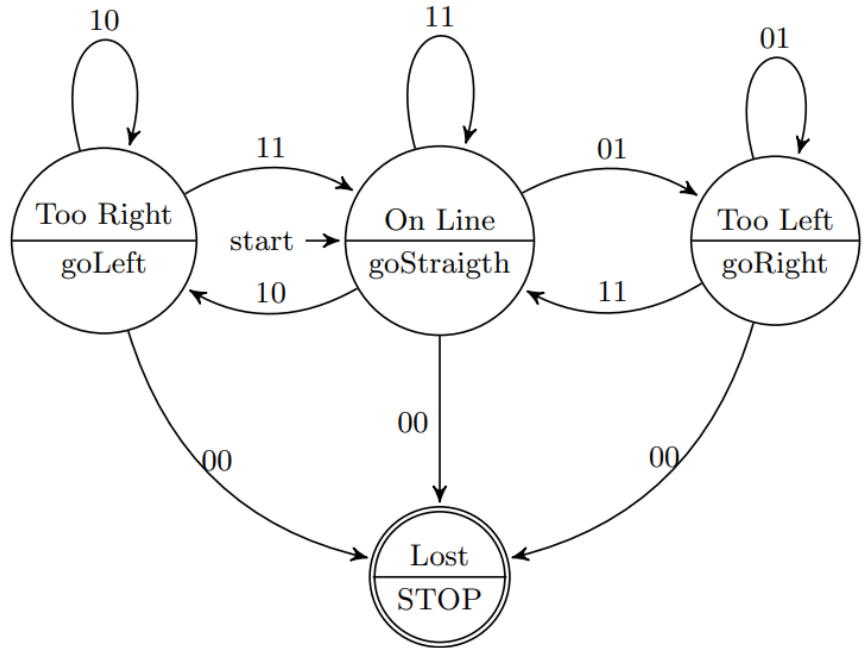
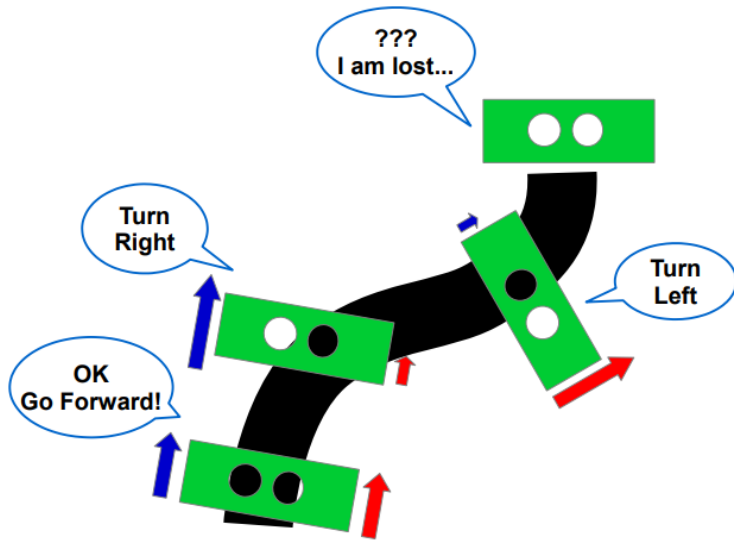






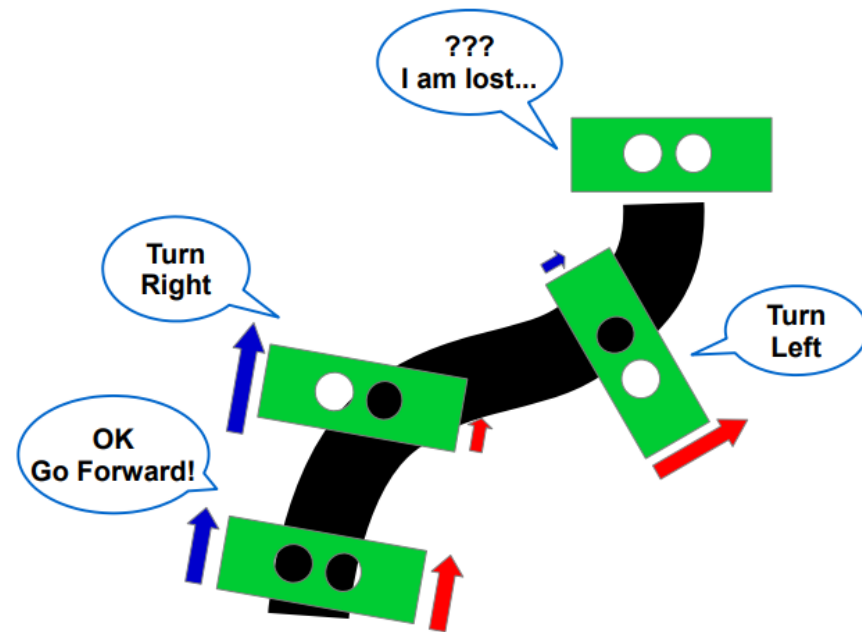


# Máquina de Estado



# Máquina de Estado

- Neste exemplo de um robô seguidor de linhas, ele tem 4 estados:
  - Virar a direita
  - Virar a esquerda
  - Linha reta
  - Parado
- Decisão é feita através do output do sensor IR





# Máquina de Estado

```
class Control():
    def __init__(self):
        self.robot_state = "procura" # Estado atual

        self.robot_machine = {
            "procura": self.procura,
            "aproxima": self.aproxima,
            "para": self.para
        }

    def procura(self) -> None:
        ...
        self.robot_state = "aproxima"

    def aproxima(self) -> None:
```

# Controle Proporcional

# Novos paradigmas



Aprendizado por Reforço

Machine Learning



Planejamento e Otimização

Modelar o ambiente em questão e otimizar a trajetória e comportamento de acordo.



Teoria de Controle

Ajuste através de métodos matemáticos.

**Exemplo: Controle Proporcional**

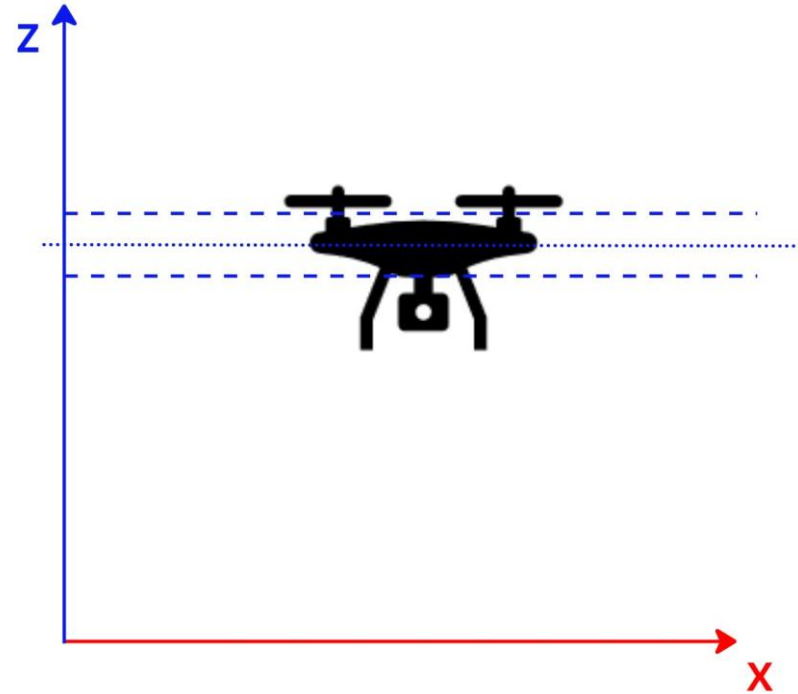


Árvore de Comportamento

Sequência de decisões

# Controle Proporcional

- Na teoria de controle, controle proporcional é uma forma de controle onde a ação, output, do sistema é proporcional ao erro entre o ponto atual e o ponto desejado.
- Exemplo:  
<http://grauonline.de/alexwww/ardumower/pid/pid.html>
- $u(t) = K_p * e(t)$



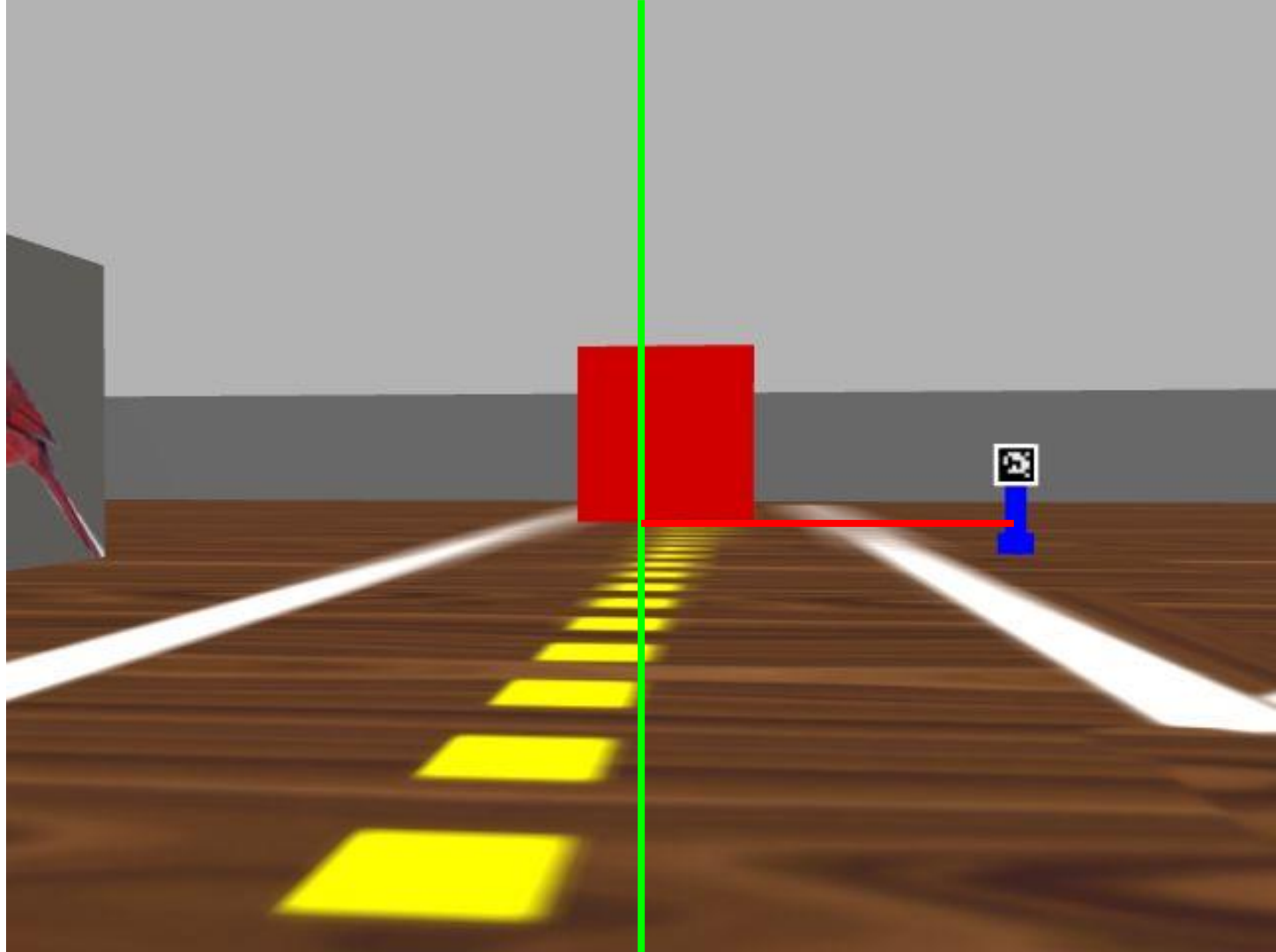


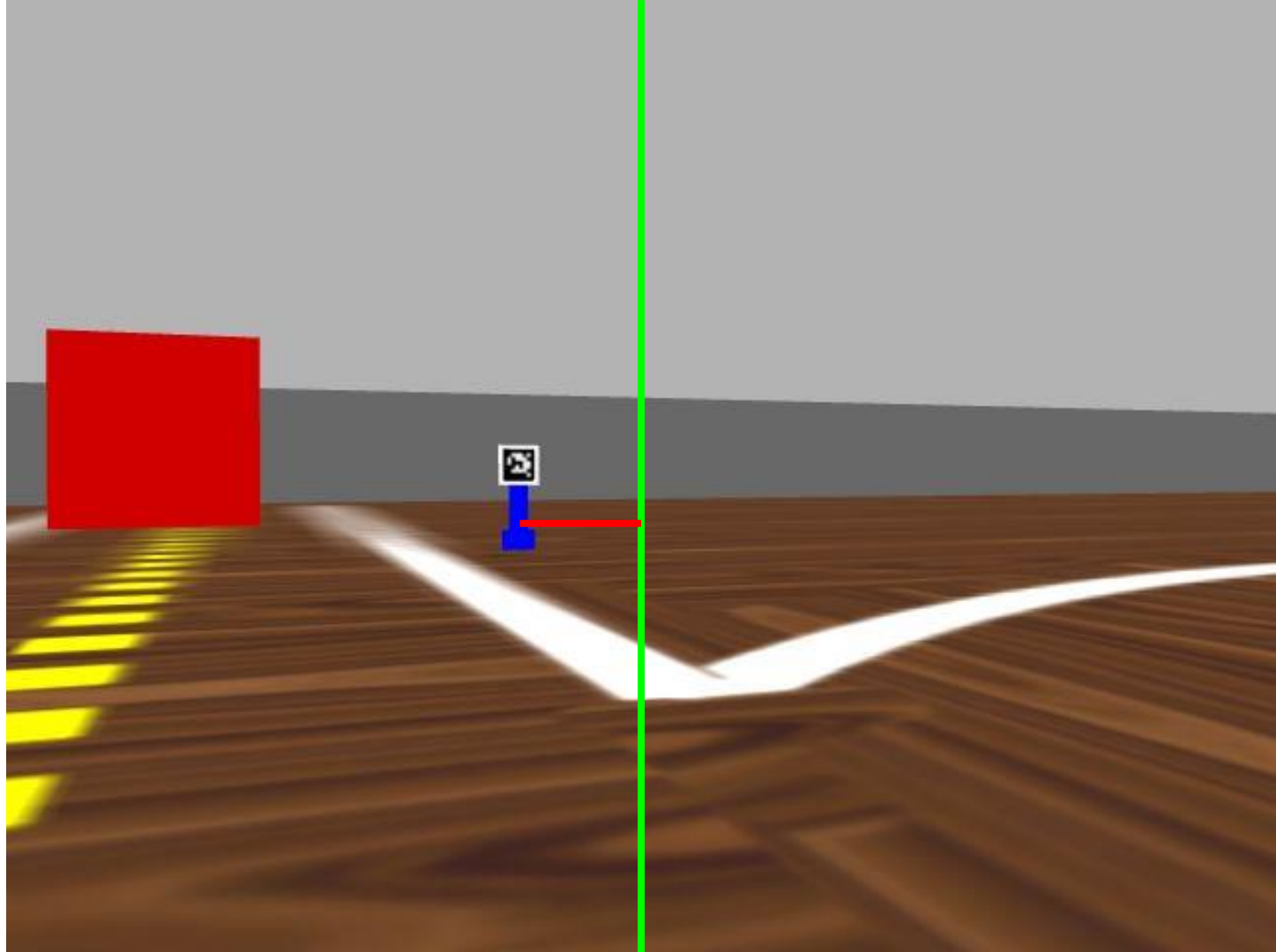
# Controle Proporcional

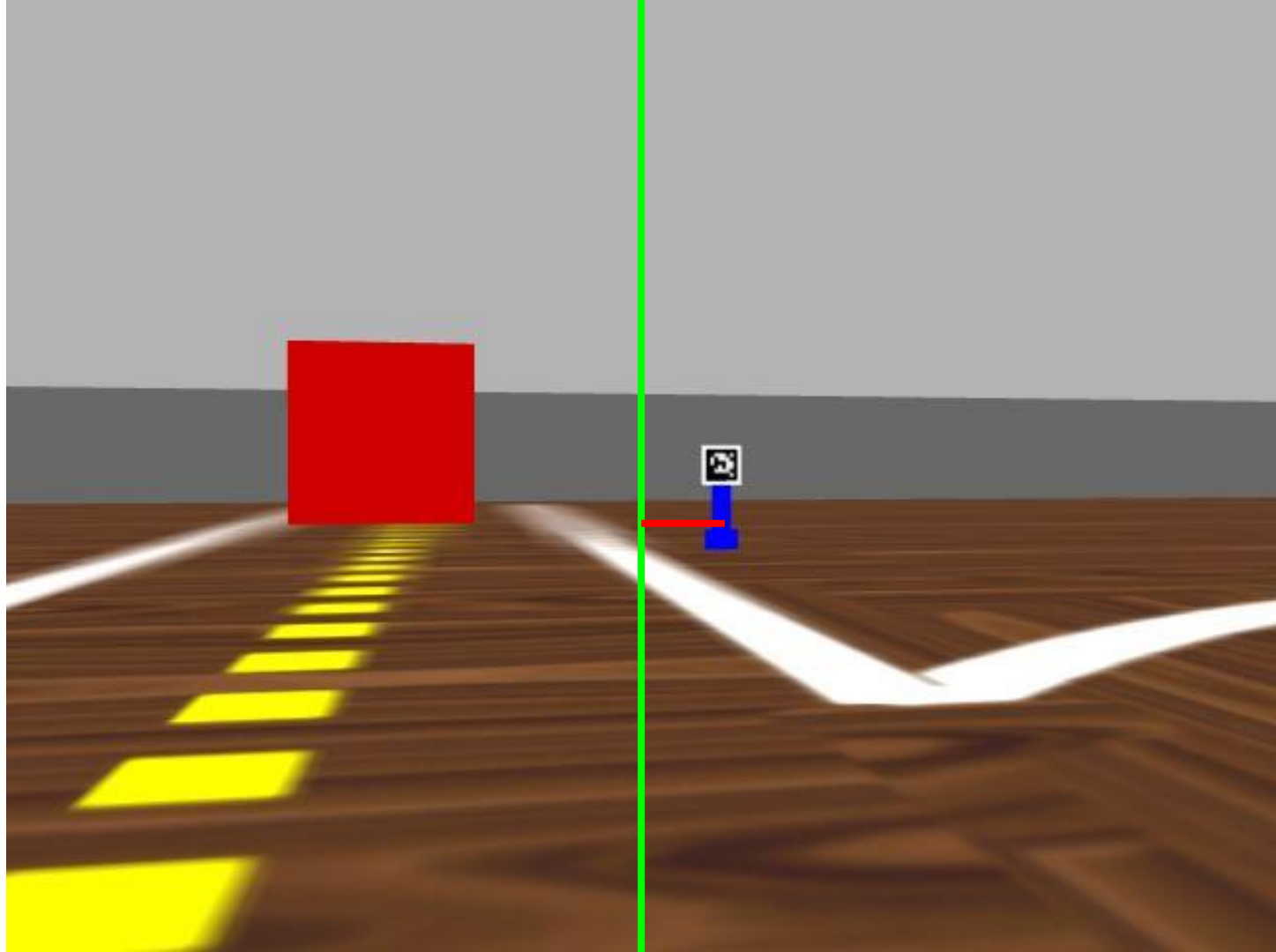
$$u(t) = K_p * e(t)$$

```
class Follower:
    def __init__(self):
        self.cx = -1
        self.kp = 100
        ...
    def image_callback(self, msg):
        ...
        self.w = image.shape[2]
        ...
        self.cx = int(M['m10']/M['m00'])

    def control(self):
        err = self.w/2 - self.cx # e(t)
        self.twist.angular.z = float(err) / self.kp
```





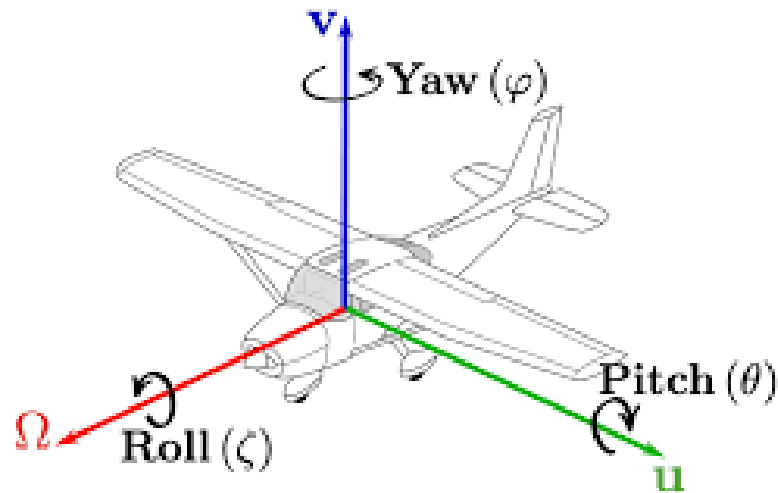




# Localização

# Localização

- O robô consegue interagir melhor com o ambiente se ele sabe onde está. Pense mundo como um mapa em 2D com vista superior. Então a localização do robô, é as coordenadas (x,y).
  - Robô Simulado: Localização absoluta.
  - Robô Real: Localização com base no encoder das rodas.
- Além de localização, o robô tem uma orientação. Orientação podem ser expressas de duas formas:
  - Euler (3-upla)
    - Combinação de rotação:
      - $R = R_x * R_y * R_z$
  - Quaternions (4-upla)
    - Números imaginários



# Odometria

Tópico: /odom

```
def odom_callback(self, data: Odometry):
    self.position = data.pose.pose.position

    orientation_list = [data.pose.pose.orientation.x,
                        data.pose.pose.orientation.y,
                        data.pose.pose.orientation.z,
                        data.pose.pose.orientation.w]

    self.roll, self.pitch, self.yaw = euler_from_quaternion(orientation_list)

    # converter o angulo yaw de [-pi, pi] para [0, 2pi]
    self.yaw = self.yaw % (2*np.pi)
```

# Atividades

Agora estão prontos para seguir com a APS4