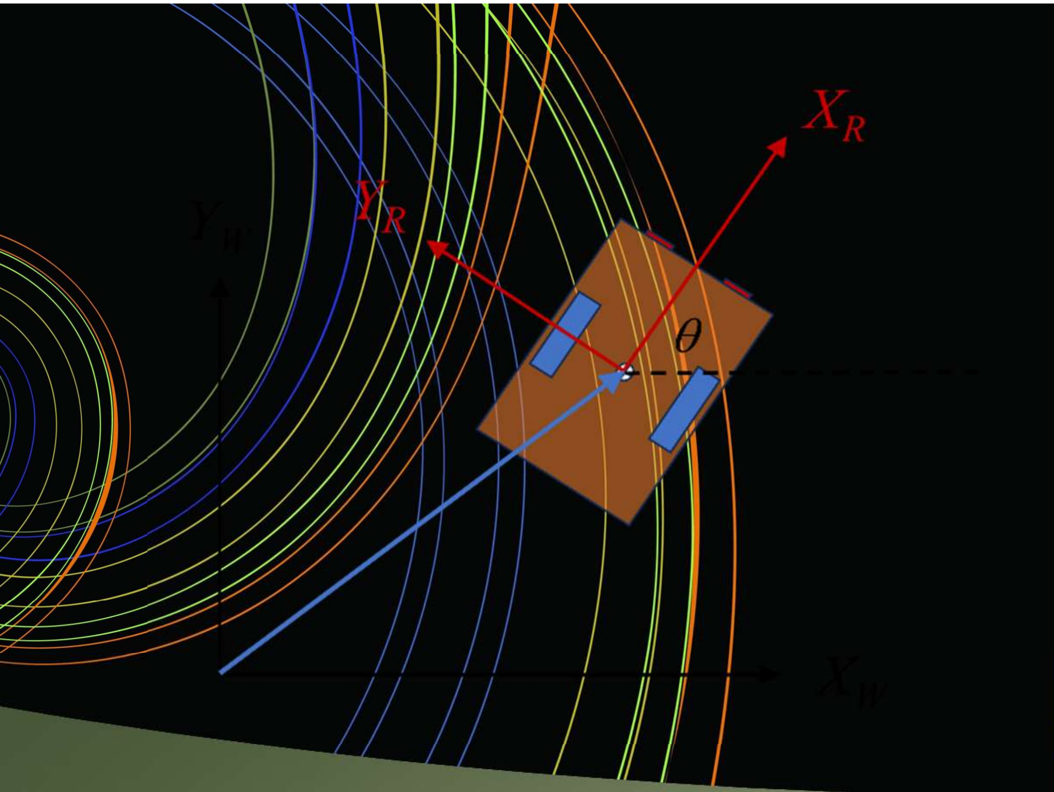
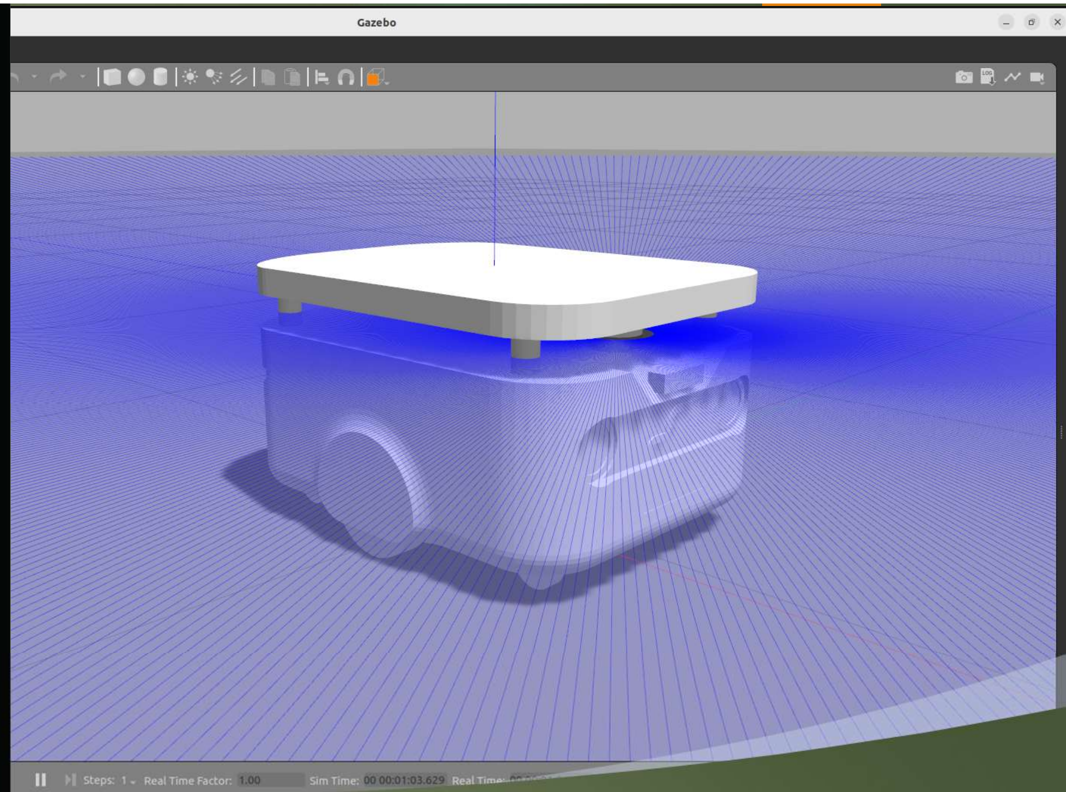
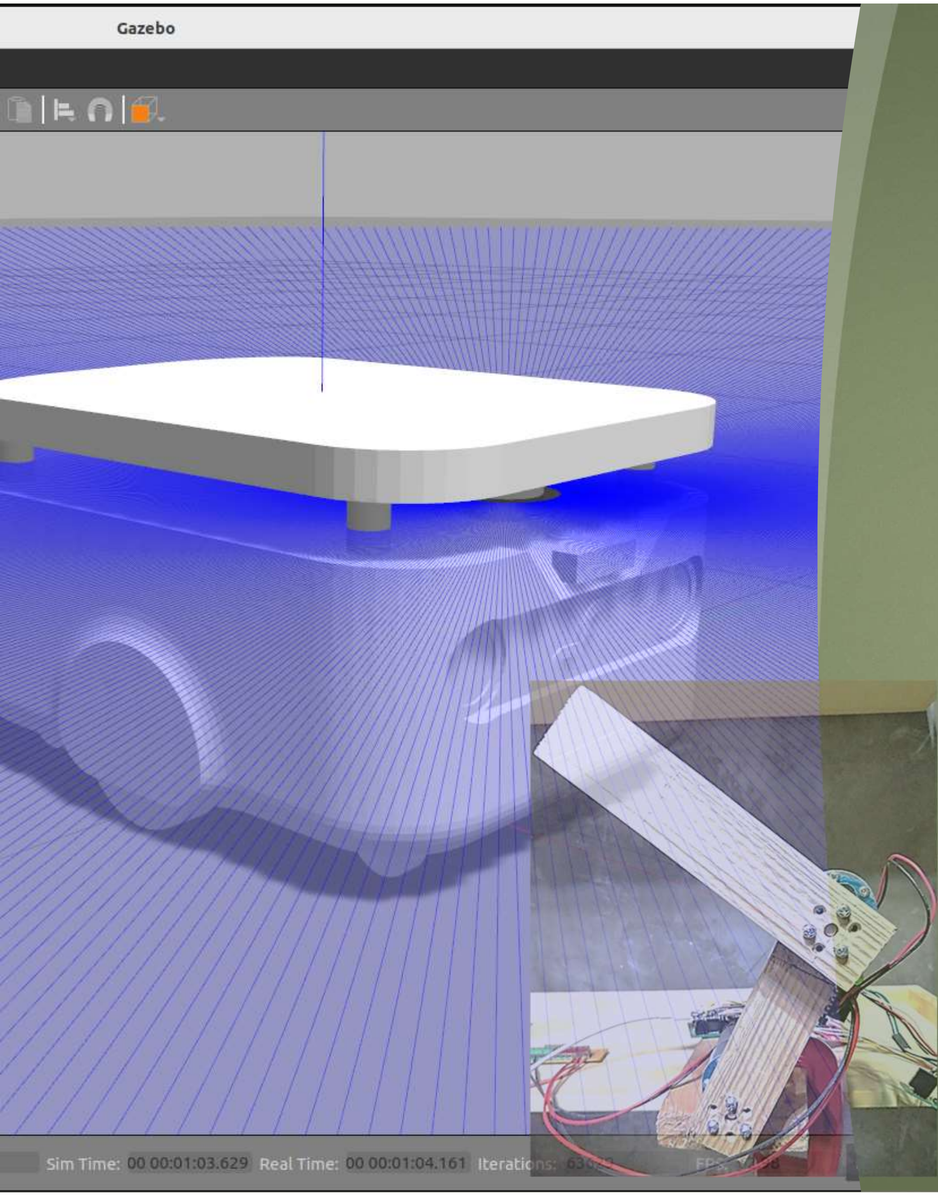




Mobile Robots

Dr.Varodom Toochinda
Dept. of Mechanical Engineering
Kasetsart University

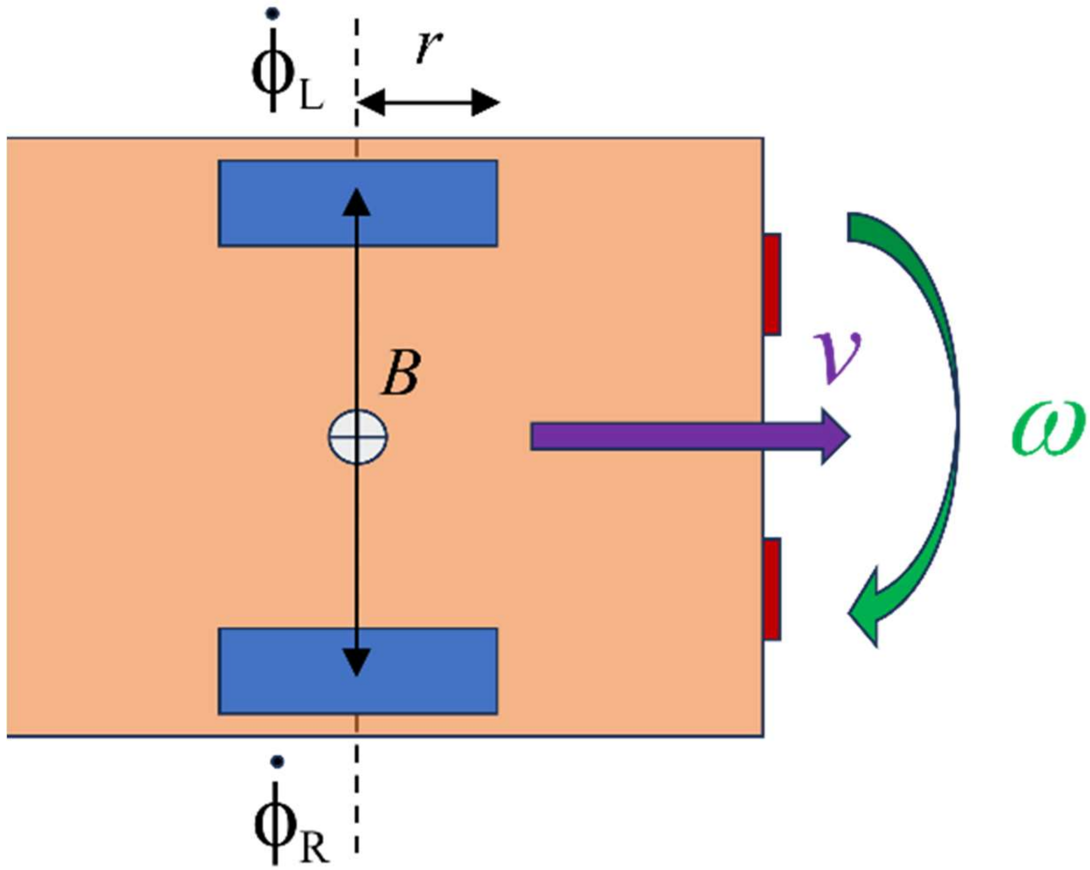
Mobile Robot



- Mobile robot controllers
  - 2 wheel differential drive robot
  - Kinematics study
  - Odometry
  - Path planning with RL (value iteration)

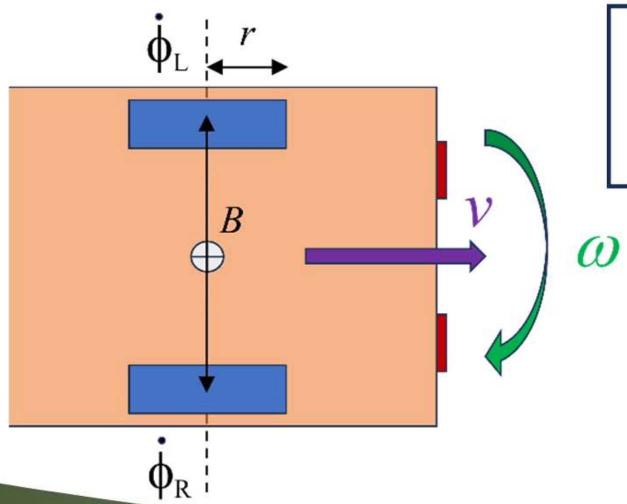
Example of 2-wheel differential drive robot

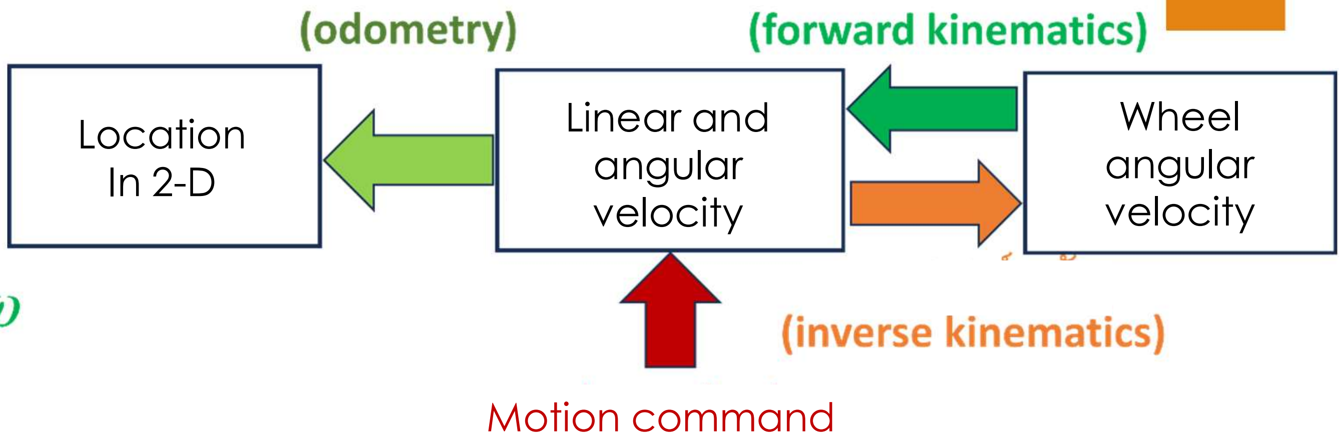
"coconut robot" by CoXsys Robotics
https://youtu.be/PkcSTzwtf5M?si=83zyKQb9XiYENqe7



Orange robot

Dimensions comparable to coconut robot, only simplified (no casters)



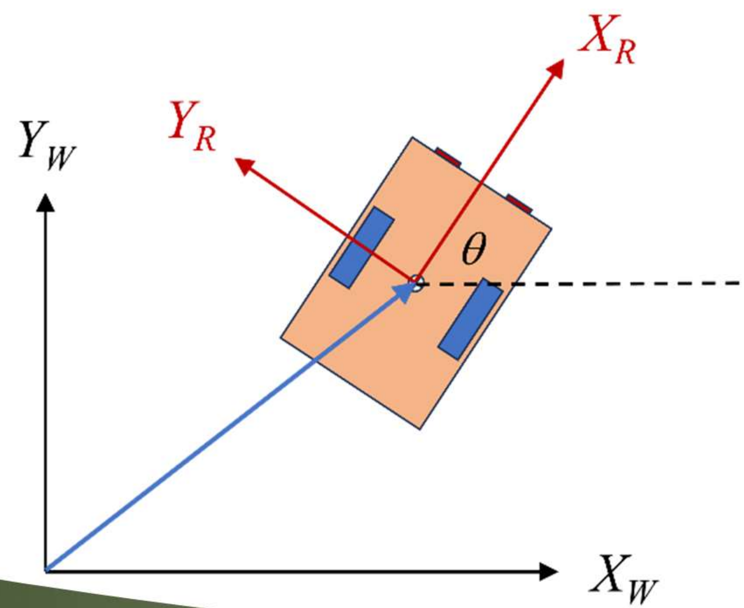


$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{r}{2}(\dot{\phi}_L + \dot{\phi}_R) \\ \frac{r}{B}(\dot{\phi}_L - \dot{\phi}_R) \end{bmatrix}$$

Forward kinematics

Velocity kinematics

Exercise : compute inverse kinematics



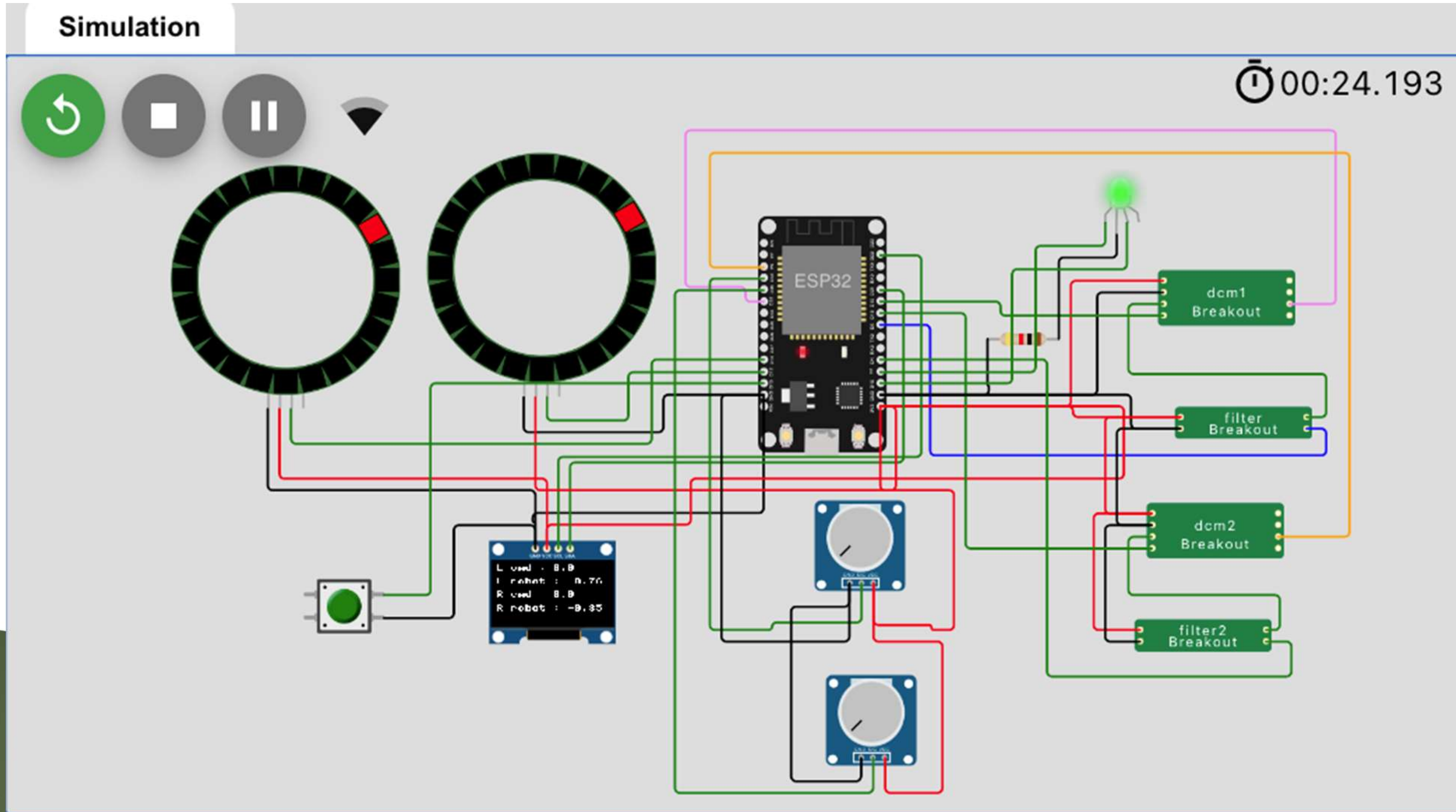
$$\begin{bmatrix} {}^{W}v_X \\ {}^{W}v_Y \\ {}^{W}\omega_Z \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{bmatrix}$$

$$x(t) = x_0 + \int_0^t v(\tau)\cos(\theta(\tau))d\tau$$

$$y(t) = y_0 + \int_0^t v(\tau)\sin(\theta(\tau))d\tau$$

$$\theta(t) = \theta_0 + \int_0^t \omega(\tau)d\tau$$

Odometry (for location estimation)



Wokwi project for orange robot

https://wokwi.com/projects/389517381277100033

ESP32 pin assignment for orange robot

Pin	function
2	Red LED for enable/disable status
4	PWM for right wheel
5	PWM for left wheel
12	NeoPixel LED simulating right wheel
13	Toggle switch to disable/enable
14	NeoPixel LED simulating left wheel
15	Green LED shows ready status
18	Direction command for right wheel
19	Direction command for left wheel
22	I2C SCL (for SSD1306 OLED)
21	I2C SDA (for SSD1306 OLED)
32	Velocity command for left wheel
34	Linear velocity command from VR
35	Angular velocity command from VR
39	Velocity command for right wheel

```python
def update_tk():
    global updatestr
    updatestr = "{},{},{},{},{}".format(ddrobot_pose[X],
              ddrobot_pose[Y],ddrobot_pose[THETA],enable,trackmode)
    if online:
       client.publish('@msg/update', updatestr)
```

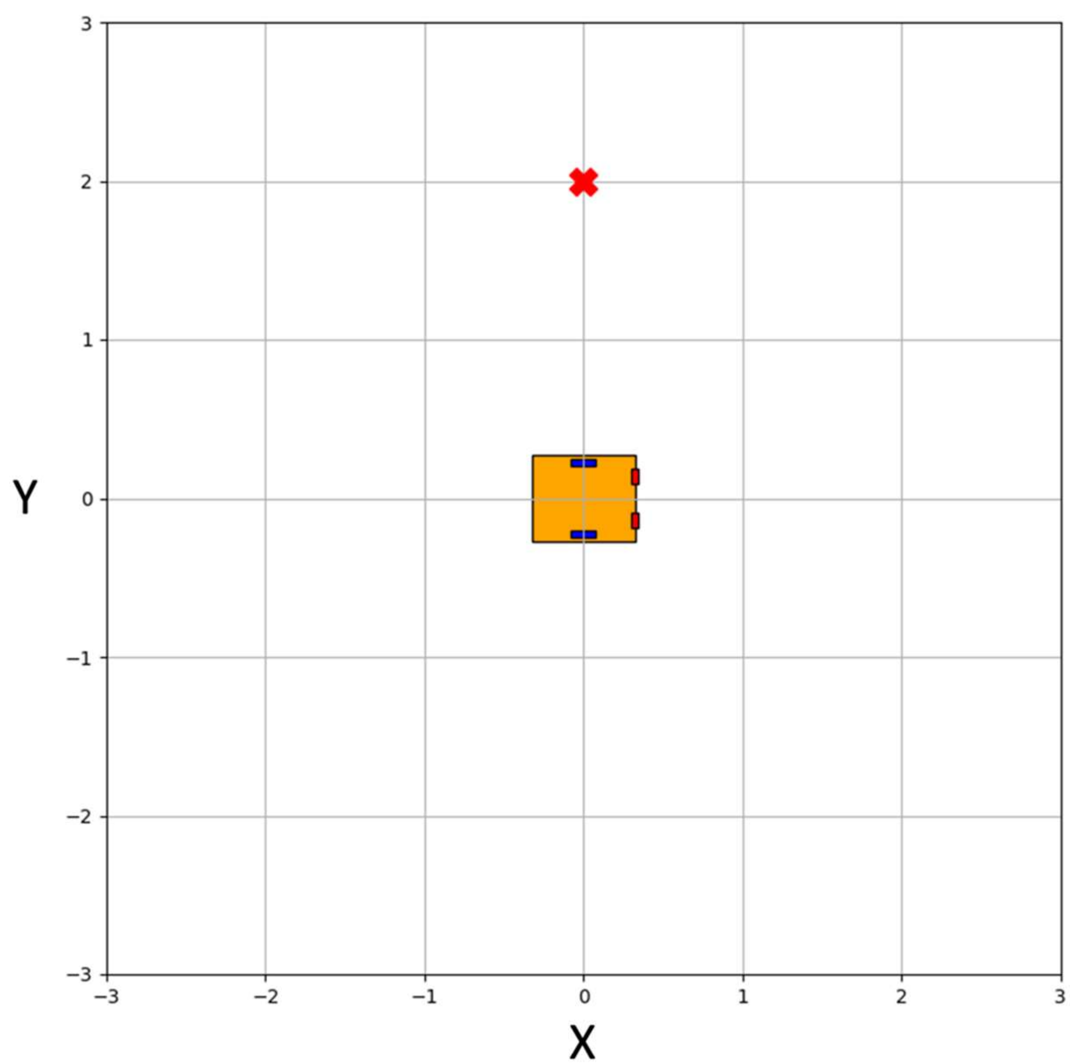
IoT or VSP communication

Wokwi link (IoT) : https://wokwi.com/projects/389518777719375873

Wokwi link (vsp) : https://wokwi.com/projects/397298628049688577

Jupyter notebook (IoT) : ddtk_track.ipynb

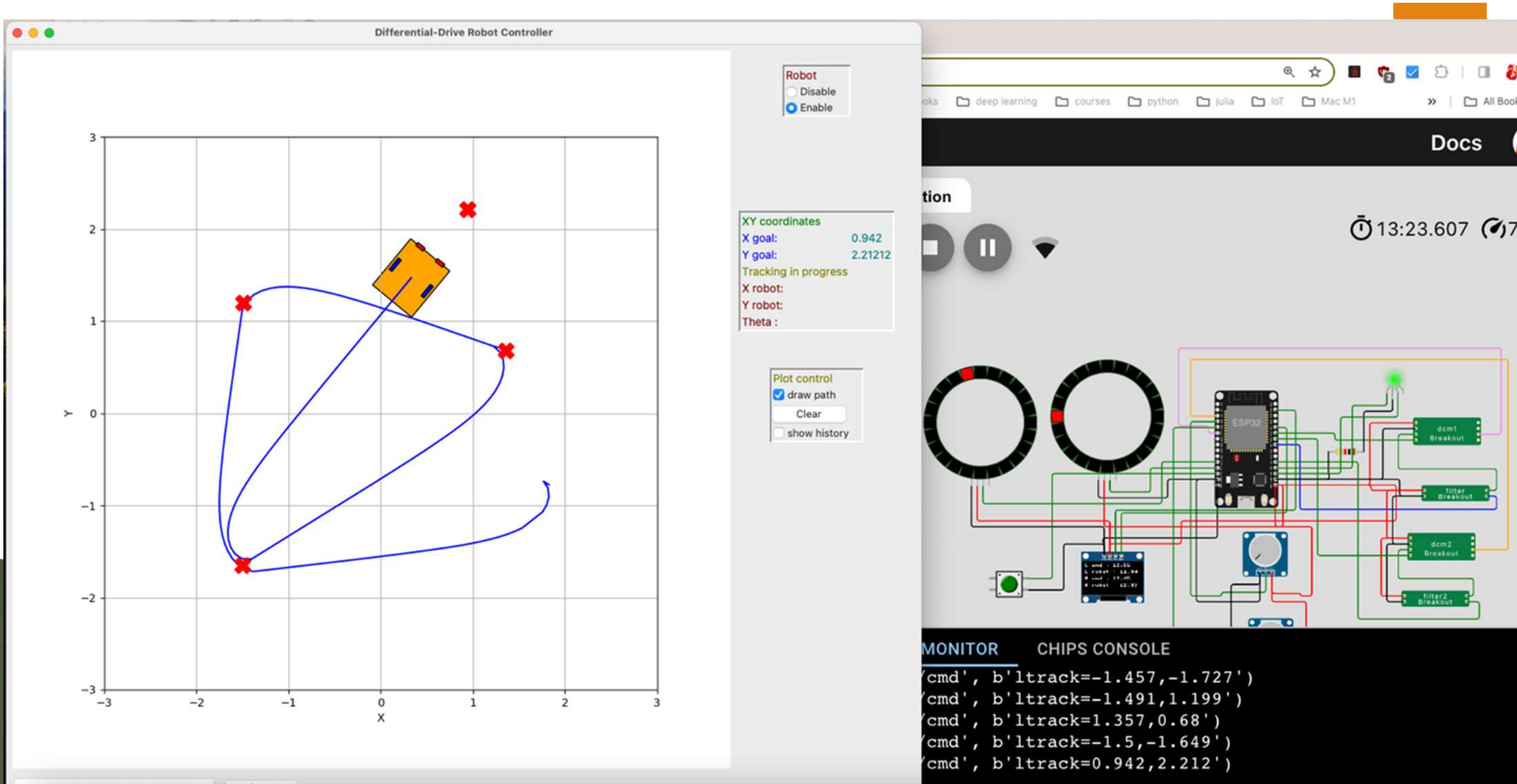
Jupyter notebook (vsp) : ddtk1_ser.ipynb



Orange robot in initial position

Motion command to target

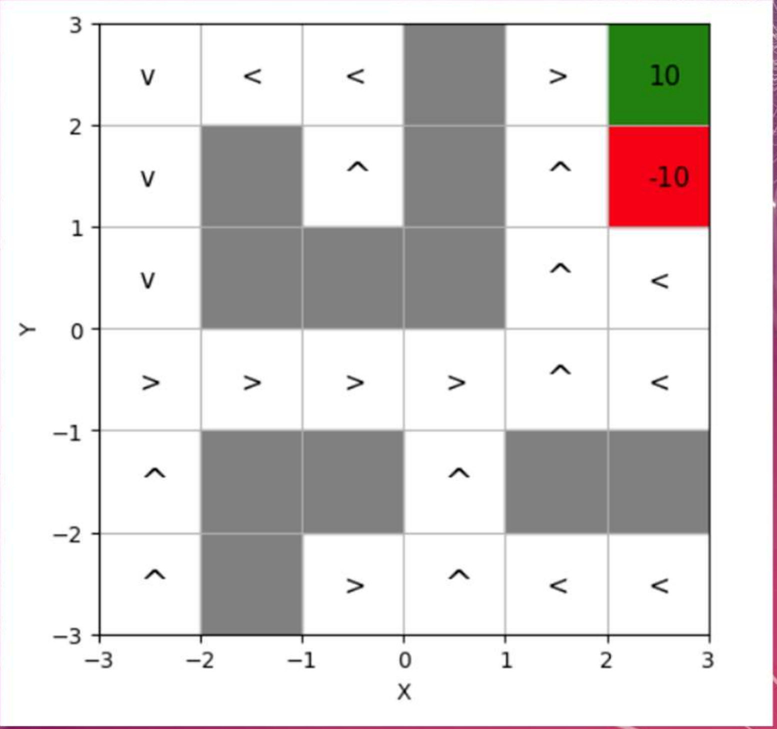
```
if trackmode:  # move to ddrobot_goal
    dp[X] = ddrobot_goal[X] - ddrobot_pose[X]
    dp[Y] = ddrobot_goal[Y] - ddrobot_pose[Y]
    norm_dp = math.sqrt(dp[X]**2 + dp[Y]**2)
    e = math.atan2(dp[Y],dp[X]) - ddrobot_pose[THETA]
    K = 3.0  # adjust this for sensitivity
    w_cmd = K*math.atan2(math.sin(e),math.cos(e))
    if norm_dp > 0.1: # distant to target is still too large
        v_cmd = 1.0
    else: # stop the robot
        v_cmd = 0.0
        w_cmd = 0.0
        trackmode = 0
        enable = 0
```



Trajectory to target example

Robot path planning

Value iteration



- An introduction to reinforcement learning
- Called dynamic programming using Bellman equation
- 6x6 gridworld can be extended to n x n (subject to "curse of dimensionality"
- All states are MDP (Markov decision process)

Bellman equation

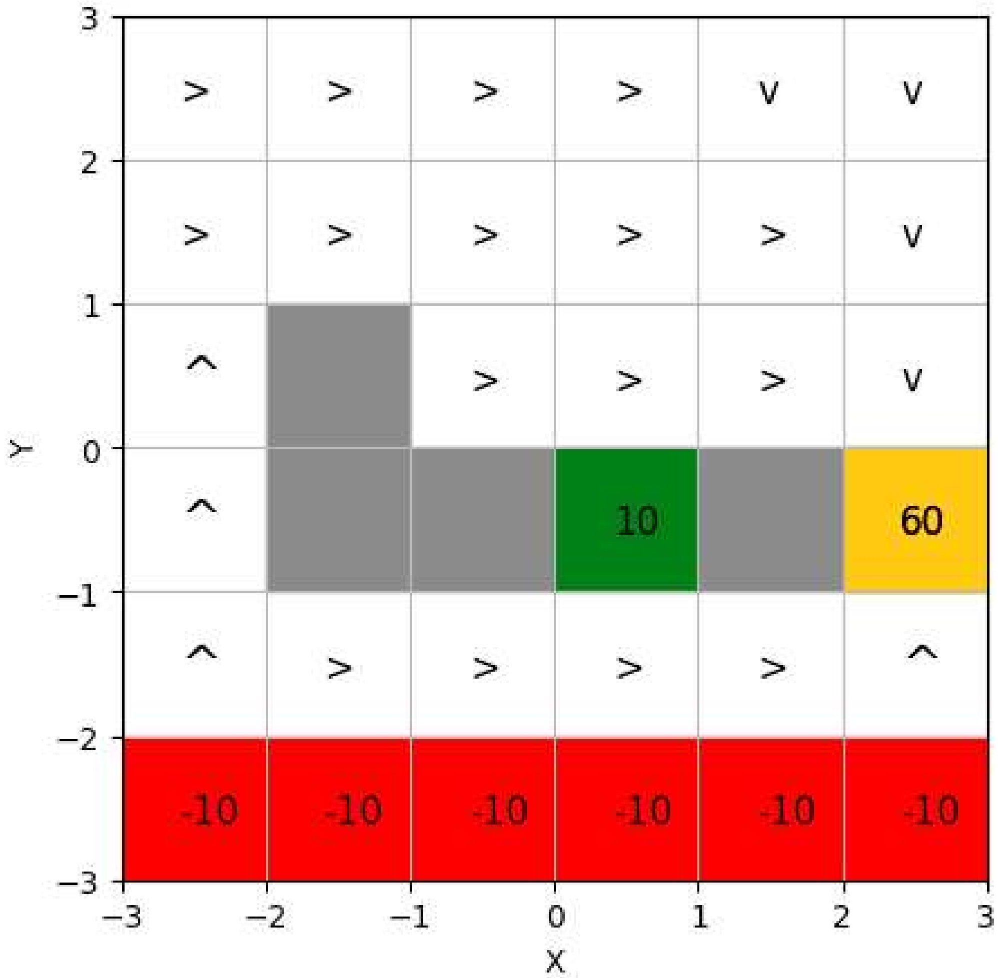
value

reward

Discount factor

Probability of making correct action

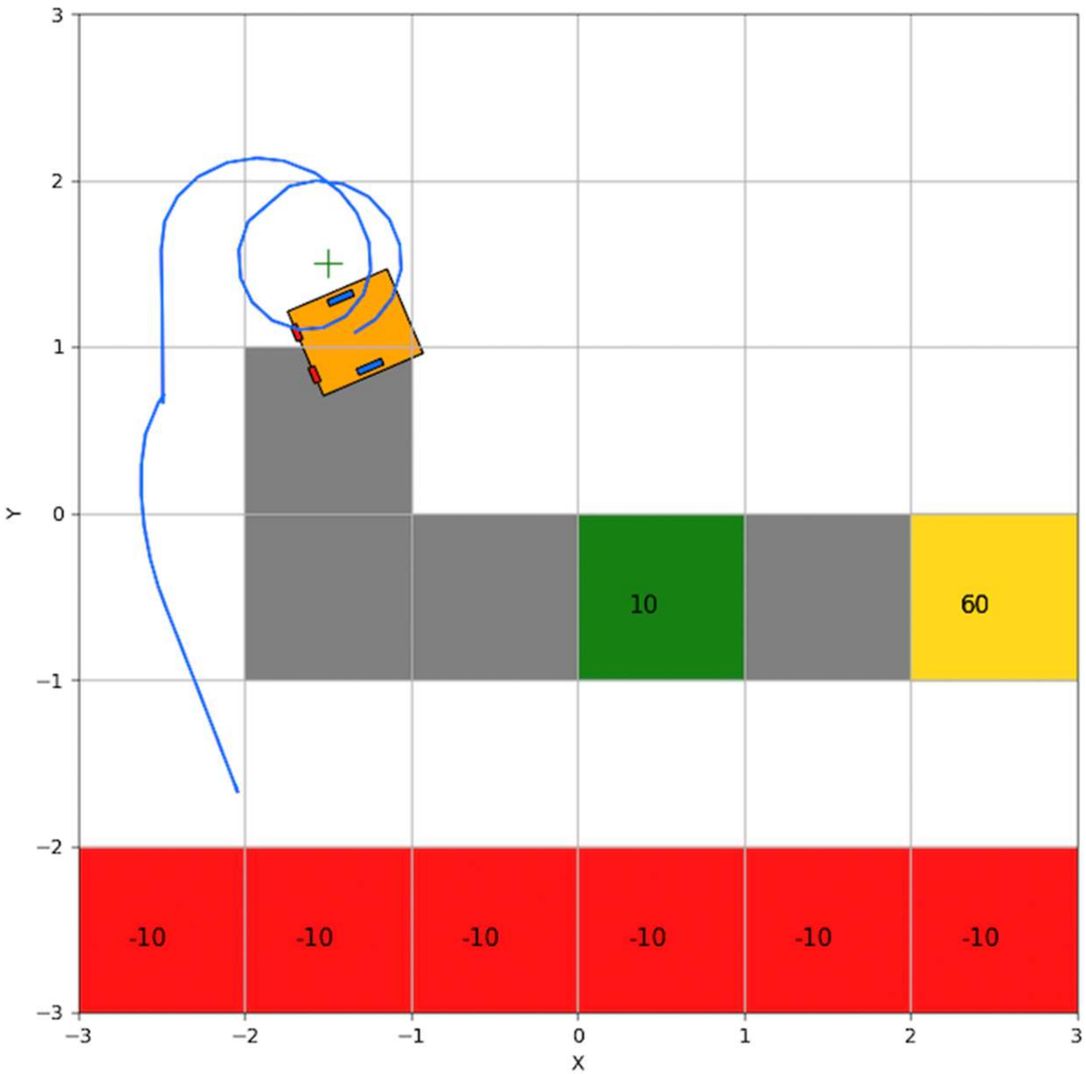
$$v(s) = r(s) + \gamma \max_{a} \sum_{s'} p(s' \mid s, a) v(s')$$



GUI for 6x6 gridworld

IoT : ddtk_gw6x6.ipynb

vsp : ddtk2_ser.ipynb

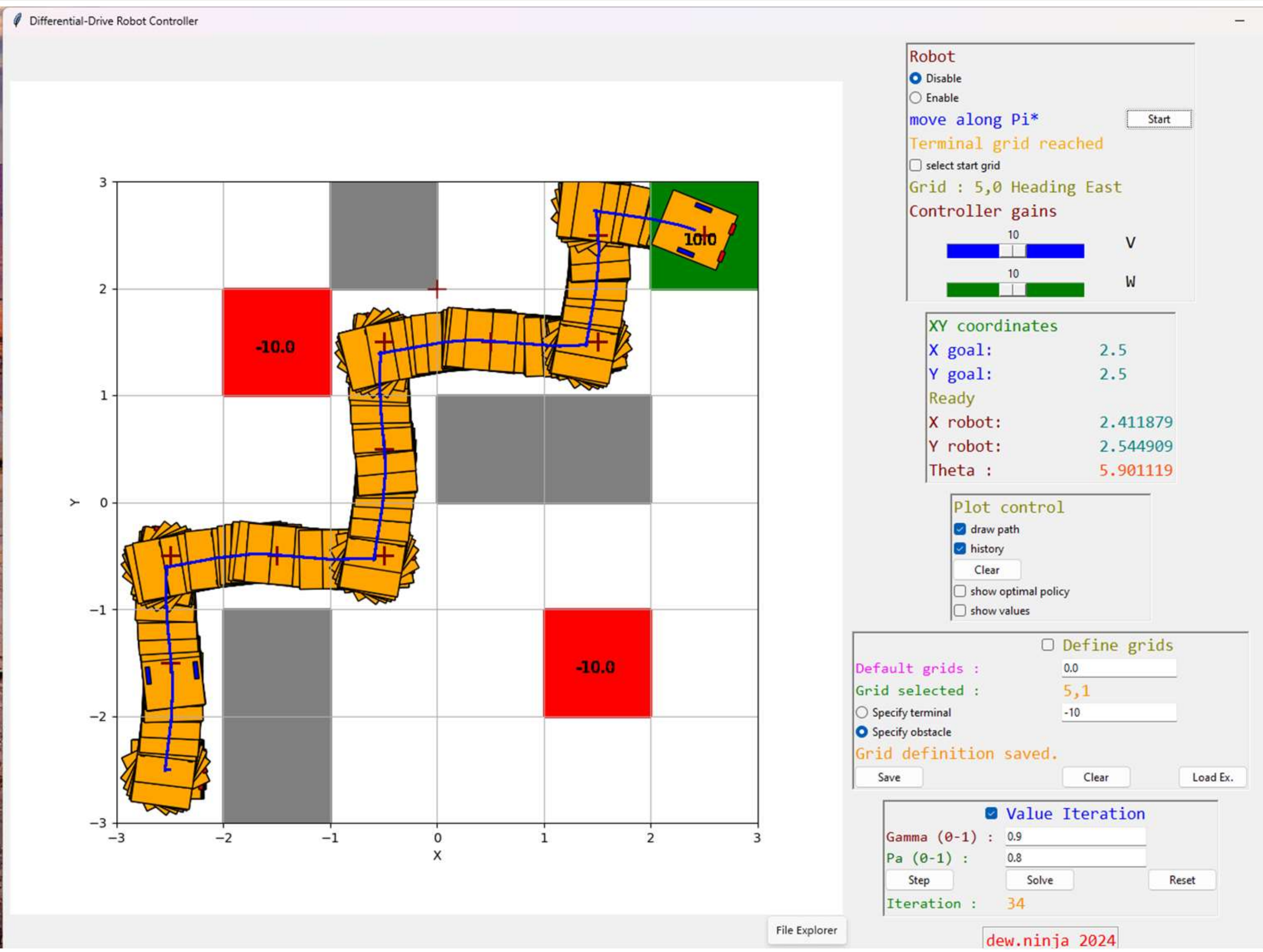


Previous motion control method is not suitable for gridworld

Exercise B.1

https://wokwi.com/projects/389594525633230849

- in Wokwi project nuws24_ex_B_1_ddrobot_iot_gw
  - Write in condition gtrackmode: so that the robot travels without hitting obstacle
  - For command gtrack in cmdInt(), change trackmode=1 to gtrackmode=1
  - If unsuccessful, use Wokwi project in the next slide



history mode feature

IoT : ddtk_gw6x6.ipynb

vsp : ddtk2_ser.ipynb

IoT : https://wokwi.com/projects/389594465354786817

vsp : https://wokwi.com/projects/397732021696556033

Path planning from optimal policy

- Specify target xg, yg as next grid center from optimal policy
- Send command gtrack = xg,yg to robot
- Get current robot position/orientation Xt, Yt, theta to draw robot
- Check if enable = 0 means robot reaches target
- Compute next grid center xg, yg
- If the robot reaches the desired destination, the episode ends

```python
def follow_pistar(self): # steer robot along optimal policy
    if self.pistart:
        # find which state it is in
        self.robot_row, self.robot_col = self.detect_grid(self.xt,
                                                          self.yt)
        self.pistarmsg_txt.set("Current grid : (" +
                     str(self.robot_row)+","+str(self.robot_col)+")")
        if not self.enable:  # initial, or target reached
            # check if a terminal is reached
            if self.terminal[self.robot_row, self.robot_col]:
                self.pistarmsg_txt.set("Terminal grid reached")
                self.pistart = False
                self.pibutton_txt.set("Start")
                self.showtarget = False
            else:
                # find center of next grid in pi* direction
                self.locate_next_grid_center()
                self.send_cmd_enable()
                time.sleep(0.5)
                self.track_next_grid()
        else: # check if robot hits an obstacle or falls into a pit
            self.check_clearance()
```

follow_pistar() function used in animate()

```
def locate_next_grid_center(self):
    if self.pimat[self.robot_row,self.robot_col] == 0:  # north
        next_row = self.robot_row - 1
        next_col = self.robot_col
    elif self.pimat[self.robot_row,self.robot_col] == 1: # east
        next_row = self.robot_row
        next_col = self.robot_col + 1
    elif self.pimat[self.robot_row,self.robot_col] == 2: # south
        next_row = self.robot_row + 1
        next_col = self.robot_col
    elif self.pimat[self.robot_row,self.robot_col] == 3: # west
        next_row = self.robot_row
        next_col = self.robot_col - 1
    self.xg = self.grid_center_x[next_col]
    self.yg = self.grid_center_y[next_row]
    self.xg_txt.set(str(self.xg))
    self.yg_txt.set(str(self.yg))
```

locate_next_grid_center() to compute next grid center from matrix self.pimat that keeps optimal policy

```python
self.grid_bounds_x = np.array([-3.0,-2.0,-1.0,0.0,1.0,2.0,3.0])
self.grid_bounds_y = np.array([3.0,2.0,1.0,0.0,-1.0,-2.0,-3.0])

def detect_grid(self,x,y): # find grid row and column from x,y
    row = col = 0
    for i in range(len(self.grid_bounds_x)-1):
        if x>self.grid_bounds_x[i] and x<= self.grid_bounds_x[i+1]:
            col = i
    for i in range(len(self.grid_bounds_y)-1):
        if y<=self.grid_bounds_y[i] and y> self.grid_bounds_y[i+1]:
            row = i
    return row, col
```

Helper function detect_grid() to compute row and column number of gridworld from x,y coordinate

```
def track_next_grid(self): # send command to track next grid
    if self.online:
        if (self.xg_prev != self.xg) or (self.yg_prev != self.yg):
            cmd_txt = "gtrack="+str(round(self.xg,3)) + "," + \
                                  str(round(self.yg,3))
            self.client.publish("@msg/cmd",cmd_txt)
            self.xg_prev = self.xg
            self.yg_prev = self.yg
```

After compute new grid center in self.xg, self.yg,
call track_next_grid() to send grack command

Robot location/orientation/status is updated

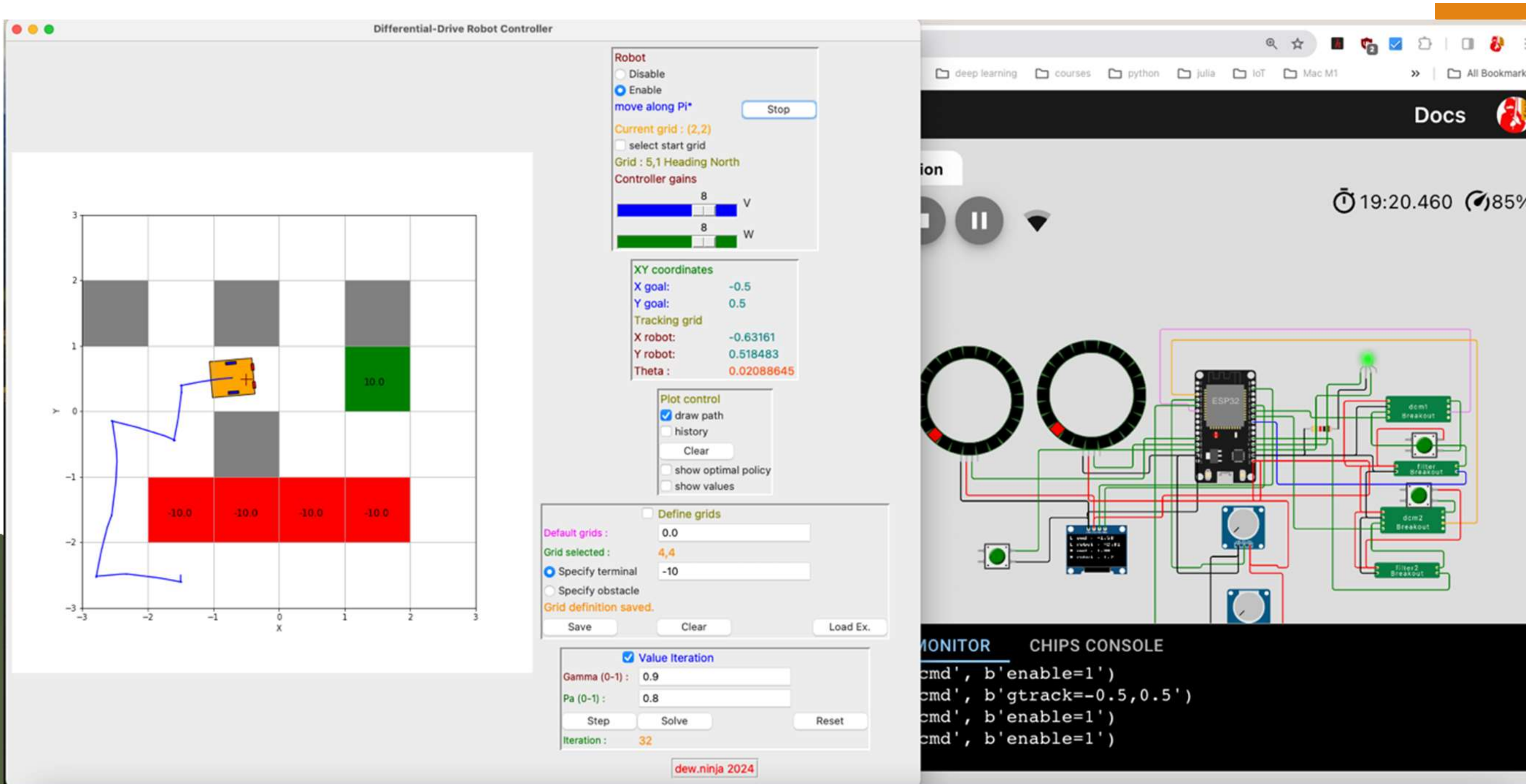
(IOT VERSION)

```
def on_message(self,client, userdata, message):
    self.rcvd_msg = str(message.payload.decode("utf-8"))
    self.rcvd_topic = message.topic

    if self.rcvd_topic == "@msg/update":
        parm_data = self.rcvd_msg.split(',')
        # strings for GUI update
        self.enable_txt.set(parm_data[3])
        self.xt_txt.set(parm_data[0])
        self.yt_txt.set(parm_data[1])
        self.theta_txt.set(parm_data[2])
        if self.gtrackmode:
            self.gtrackmsg_txt.set("Tracking grid")
        else:
            self.gtrackmsg_txt.set("Ready")
        # update parameters
        self.enable = int(parm_data[3])
        self.gtrackmode = int(parm_data[4])
        self.xt = float(parm_data[0])
        self.yt = float(parm_data[1])
        self.theta = float(parm_data[2])
```

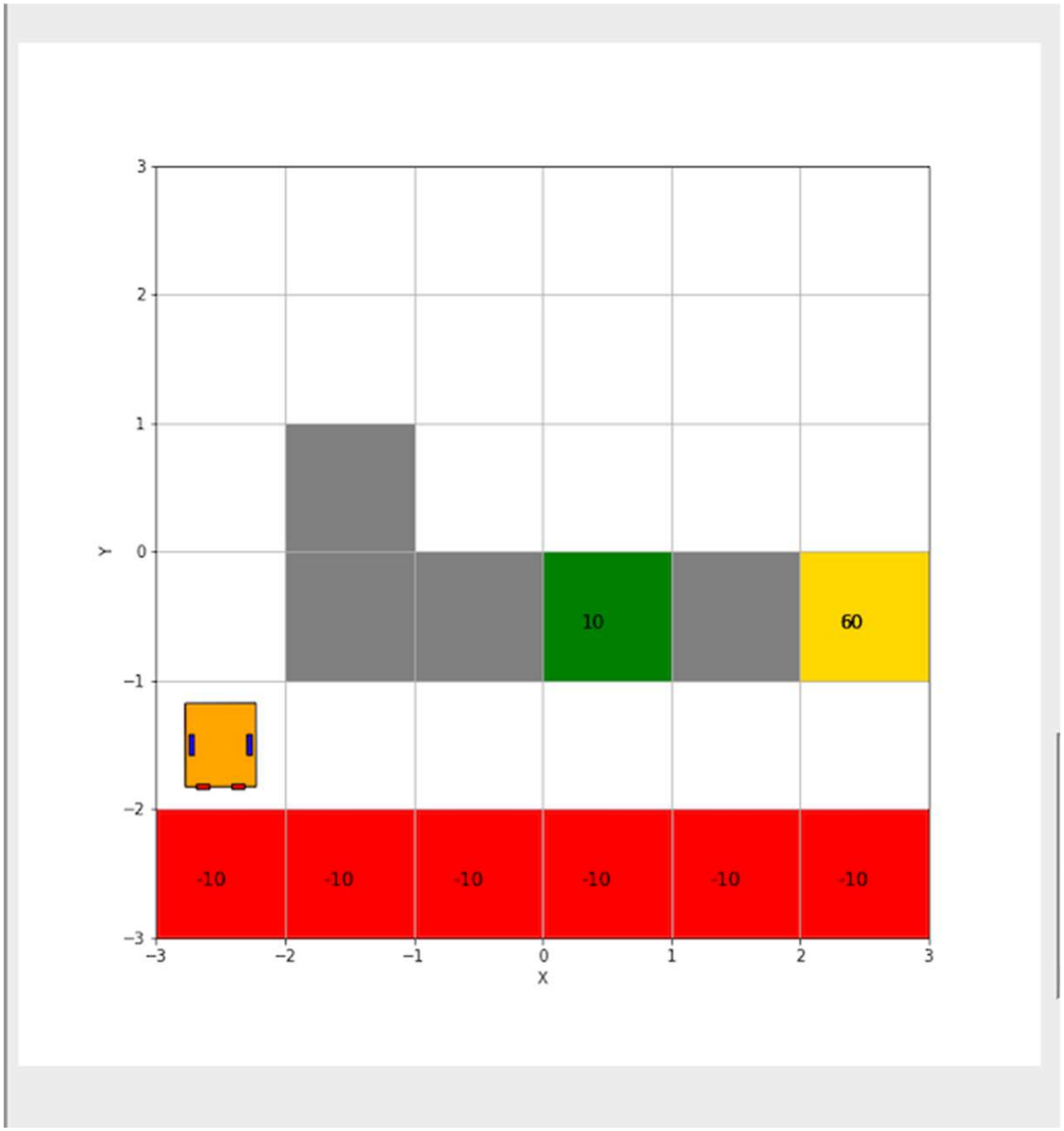
Robot location/orientation/status is updated

(vsp version)

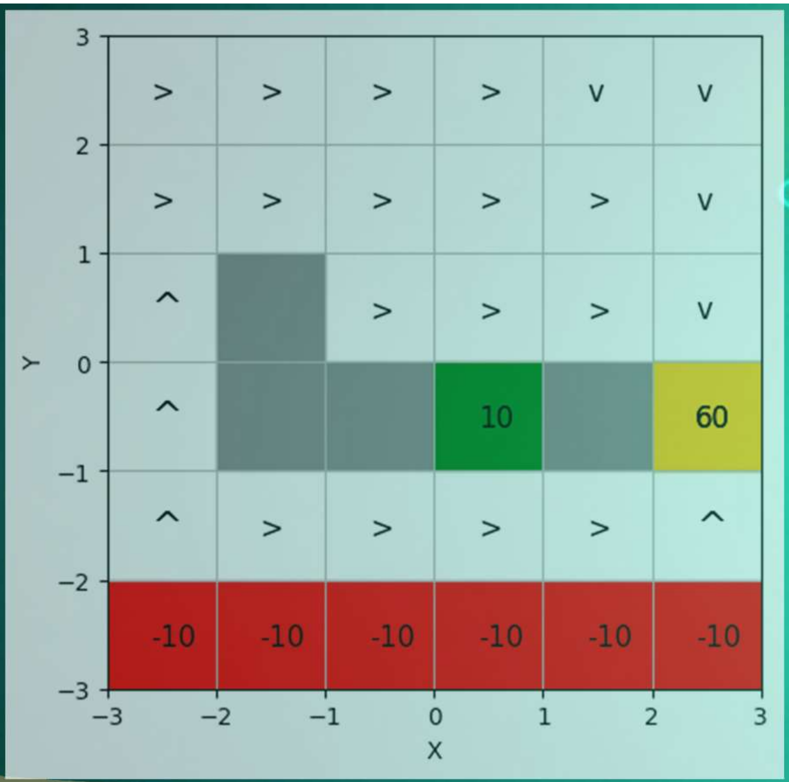
```python
if self.vspon:
    # read data from serial buffer. Then loop to put all data points to plot vectors.
    if self.streaming:
        self.msgline = self.ser.readline()
        datastr = self.msgline.decode().strip()
        if '$' not in datastr:  # not contaminated by system command/ response
            ddr_data = datastr.split(',')
            #print(ddr_data)
            #print(len(ddr_data))
            if len(ddr_data)==5:  # this prevents read error
                self.xt = float(ddr_data[0])
                self.yt = float(ddr_data[1])
                self.theta = float(ddr_data[2])
                self.enable = int(ddr_data[3])
                self.trackmode = int(ddr_data[4])

                # set text for display purpose
                self.xt_txt.set(ddr_data[0])
                self.yt_txt.set(ddr_data[1])
                self.theta_txt.set(ddr_data[2])
                self.enable_txt.set(ddr_data[3])
```



GUI in action with Wokwi simulation



Some gridworld problem example

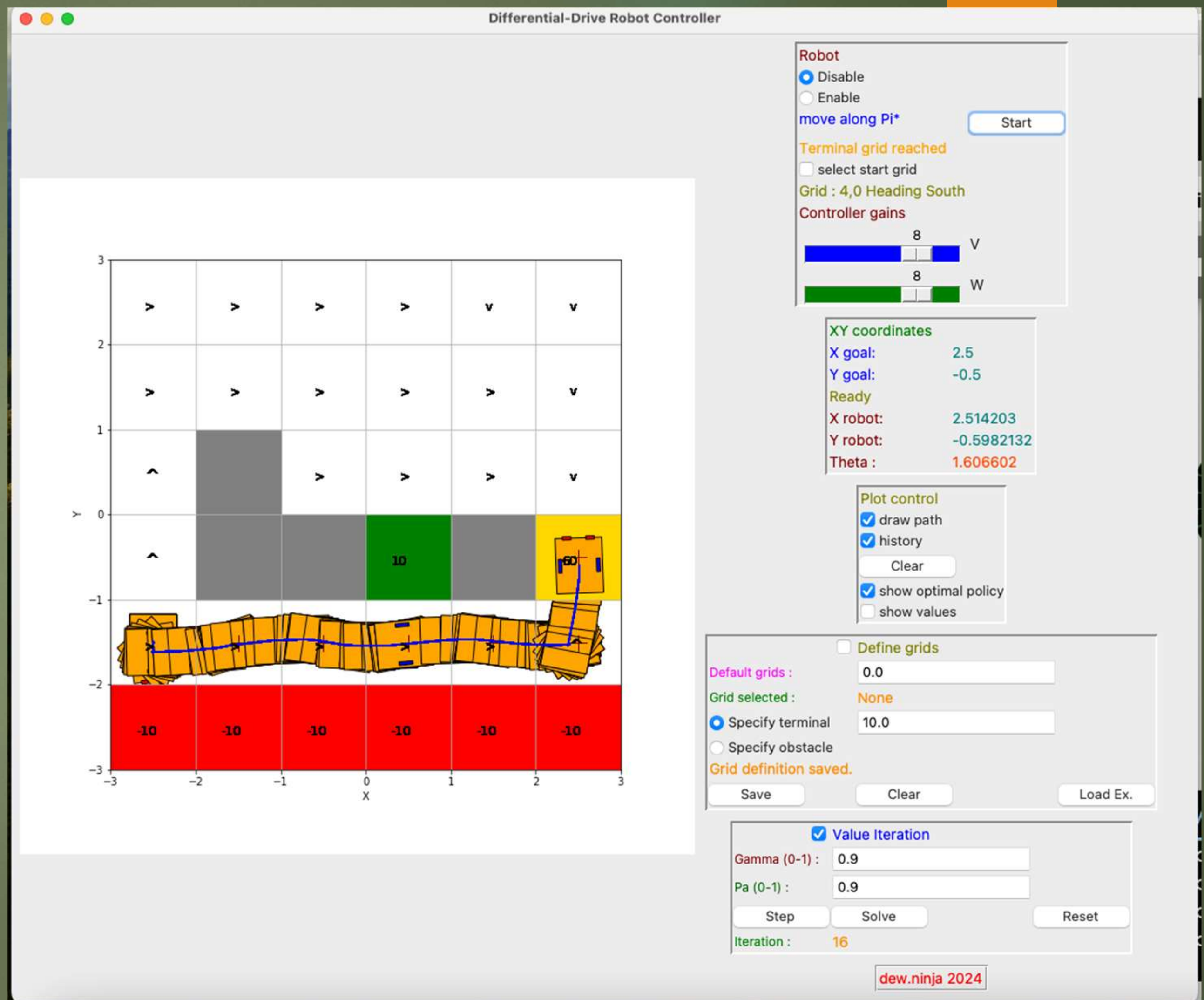


Optimal policy from value iteration

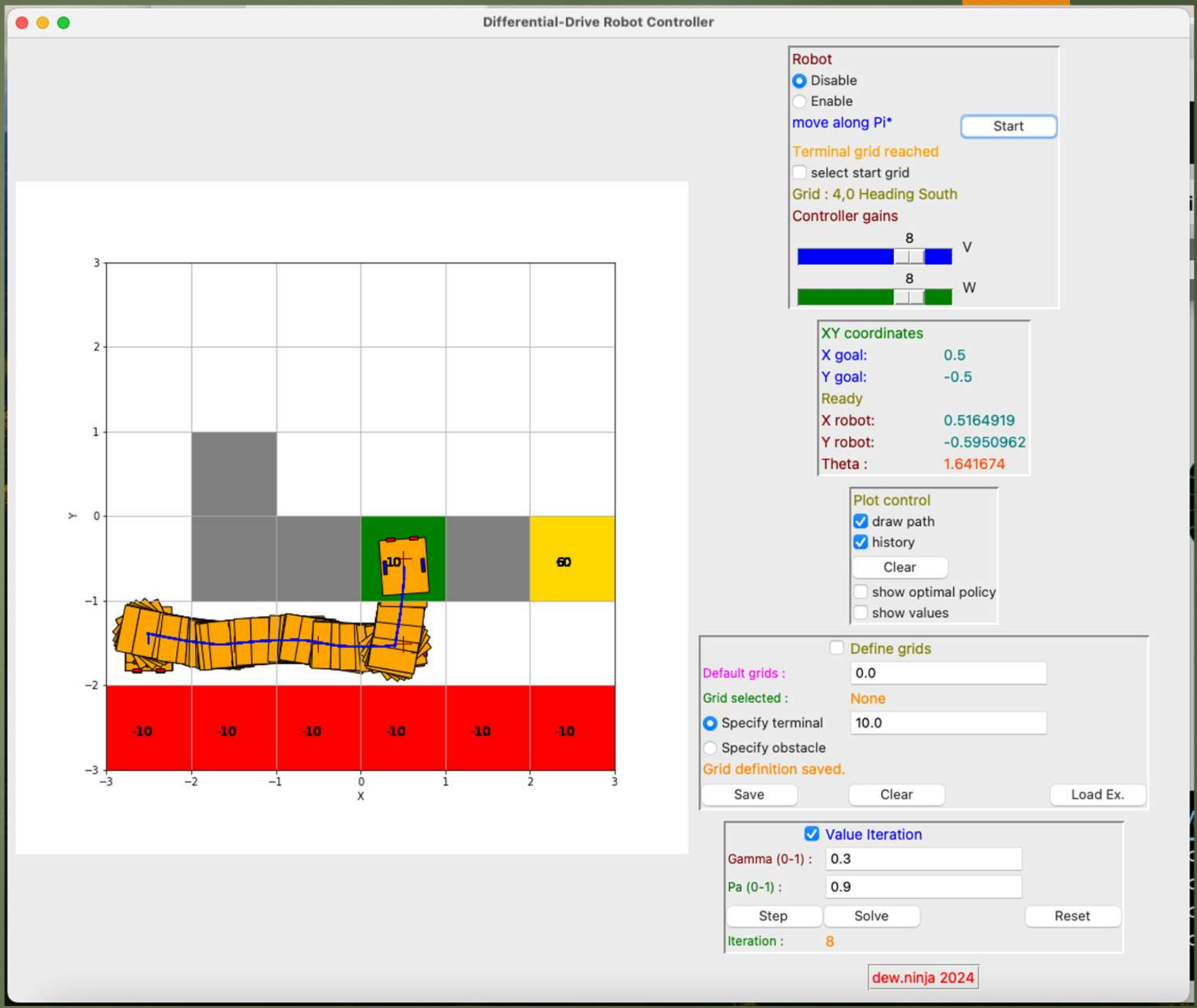
Modified from Exercise 1 in video L1 MDPs, Exact Solution Methods, Max-ent RL by Prof. Pieter Abbeel, University of California, Berkeley

https://youtu.be/2GwBez0D20A?si=XL-_Y6YStqxFA3oL

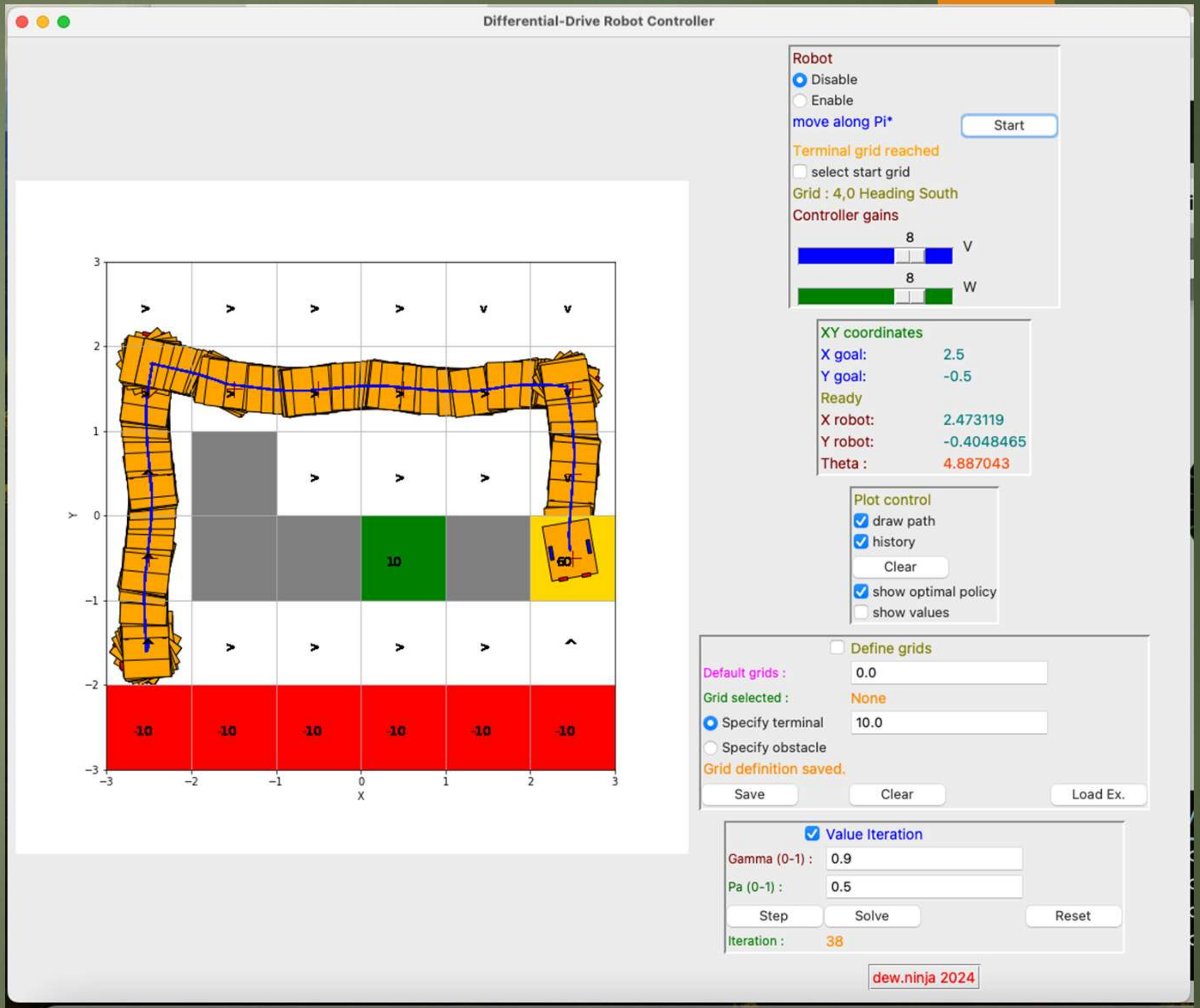
$\gamma = 0.9, P_a = 0.9$



$$\gamma = 0.3, P_a = 0.9$$

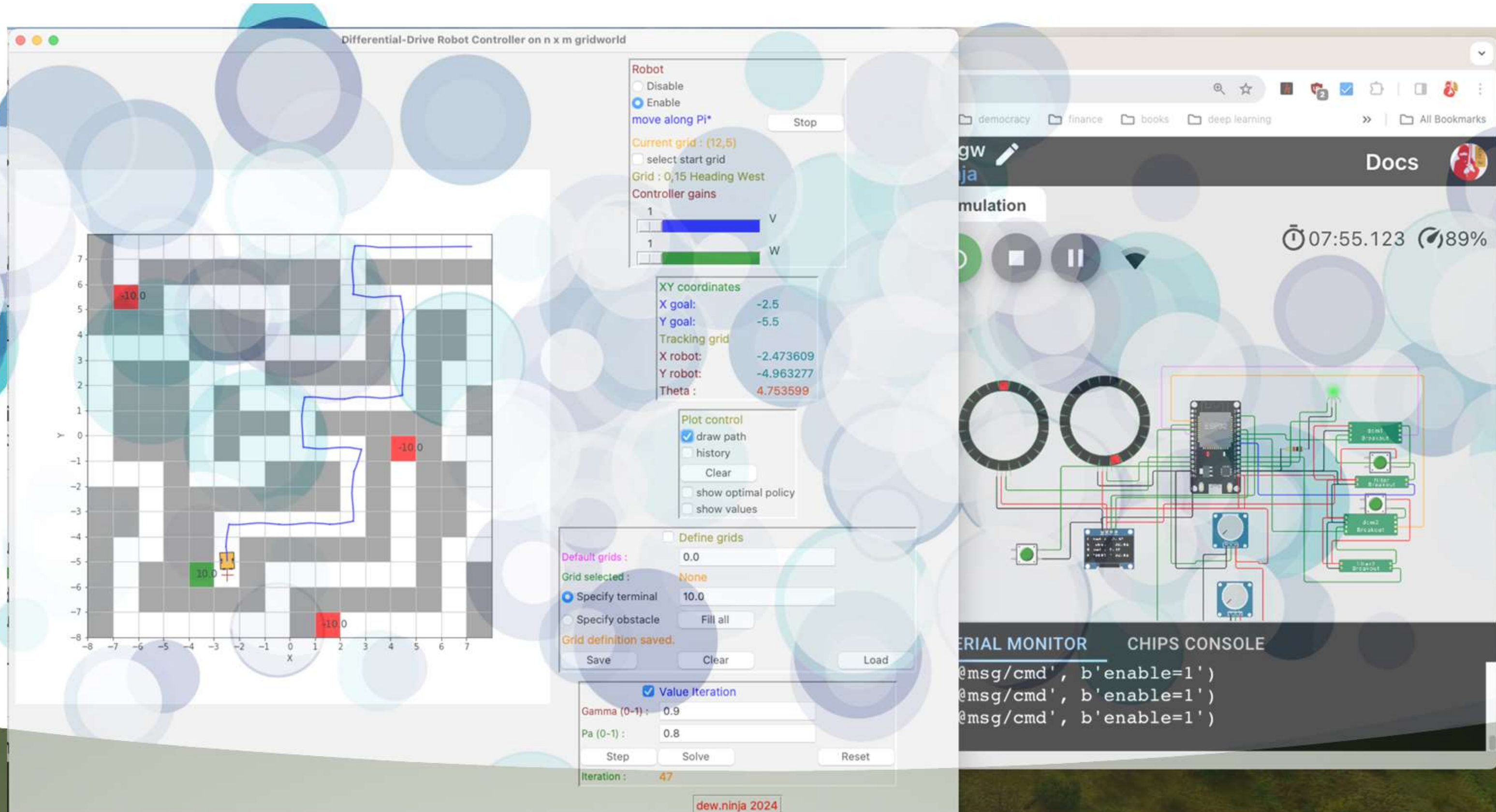


$$\gamma = 0.9, P_a = 0.5$$

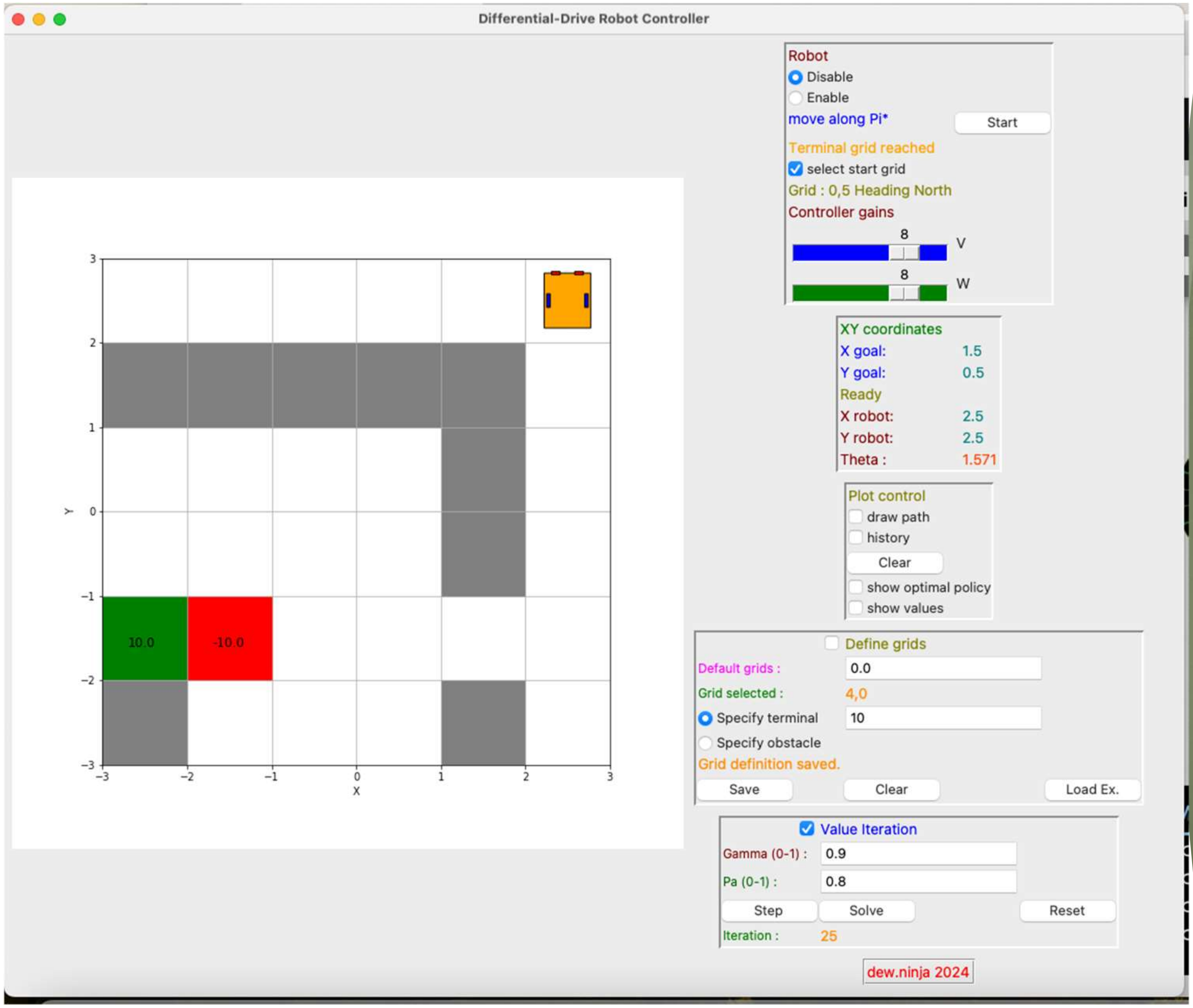


What would be the optimal policay when we select $\gamma = 0.9, P_a = 0.9$?

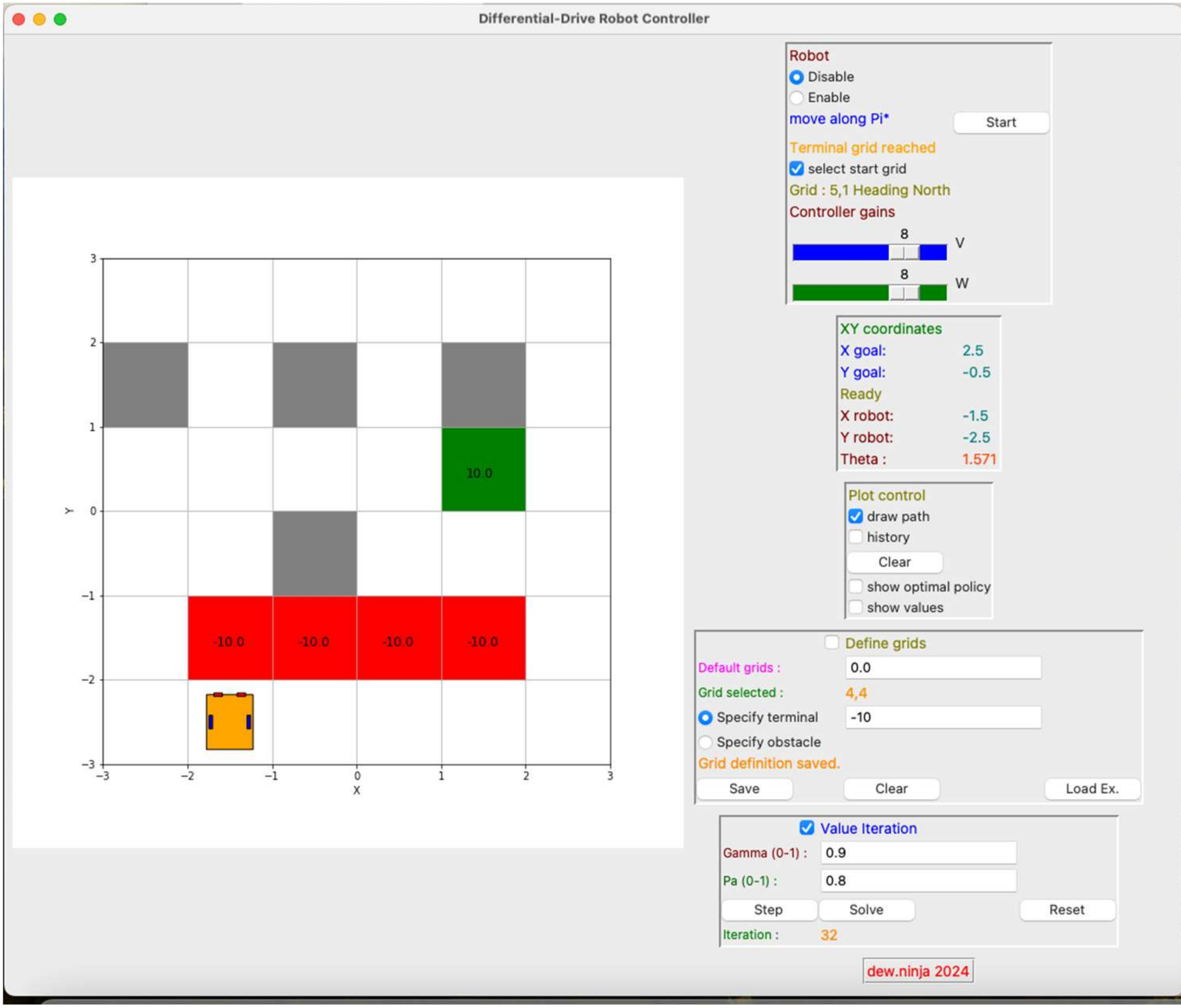
Exercise B.2



Some examples



Robot brings newspaper to room owner with a cat

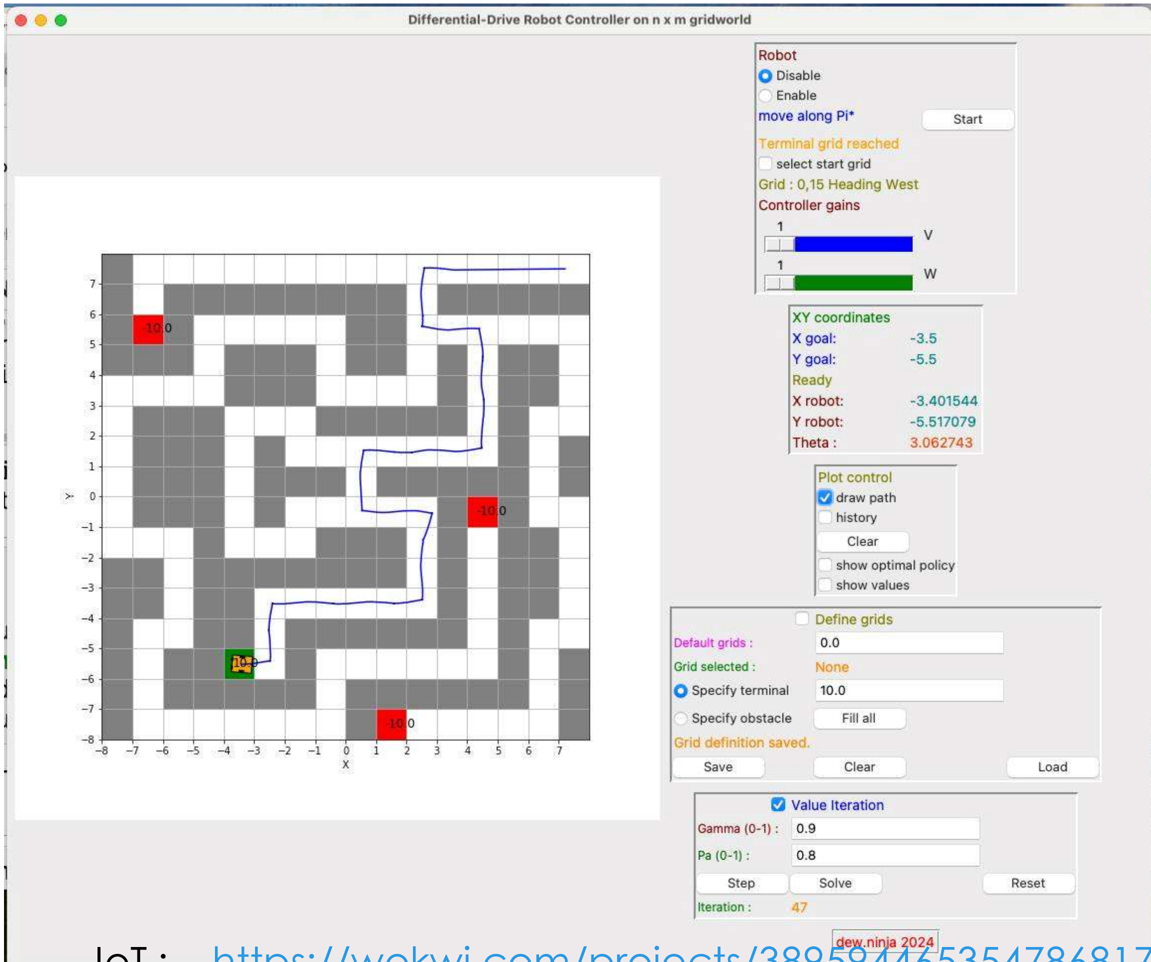


3 tables food service

Maze problem solving with nxn gridworld

IoT : ddtk_gwnxn.ipynb

vsp : ddtk3_ser.ipynb



IoT : https://wokwi.com/projects/389594465354786817

vsp : https://wokwi.com/projects/397732021696556033