DEEP REINFORCEMENT LEARNING (SLAM)

BTP PRESENTATION

Rajendra Singh
Roll No. 111601017
Computer Science and Engineering
Final Year, IIT Palakkad

TABLE OF CONTENT

1. Overview of project

Definition of the SLAM Problem

Given

- The robot's controls

$$u_{1:T} = \{u_1, u_2, u_3 \ldots, u_T\}$$

- Observations

$$z_{1:T} = \{z_1, z_2, z_3 \ldots, z_T\}$$

Wanted

- Map of the environment

$$m$$

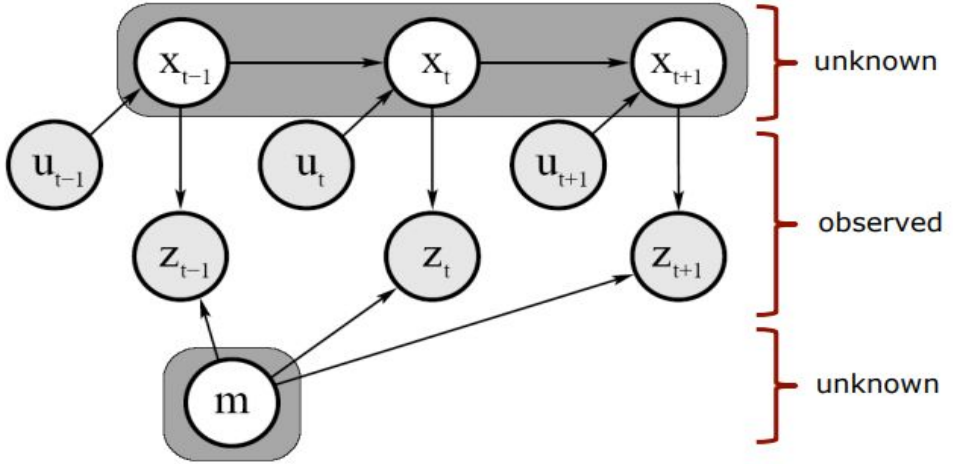
- Path of the robot

$$x_{0:T} = \{x_0, x_1, x_2 \ldots, x_T\}$$

Estimate the robot's path and the map

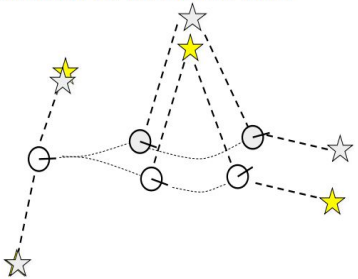
$$p(x_{0:T}, m \mid z_{1:T}, u_{1:T})$$

distribution path map given observations controls



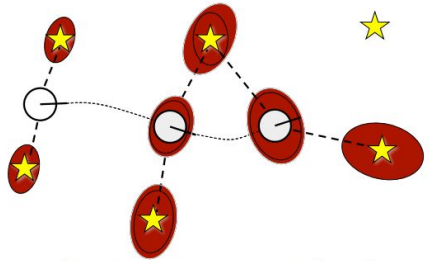
SLAM Example

- Estimate the robot's poses and the landmarks at the same time



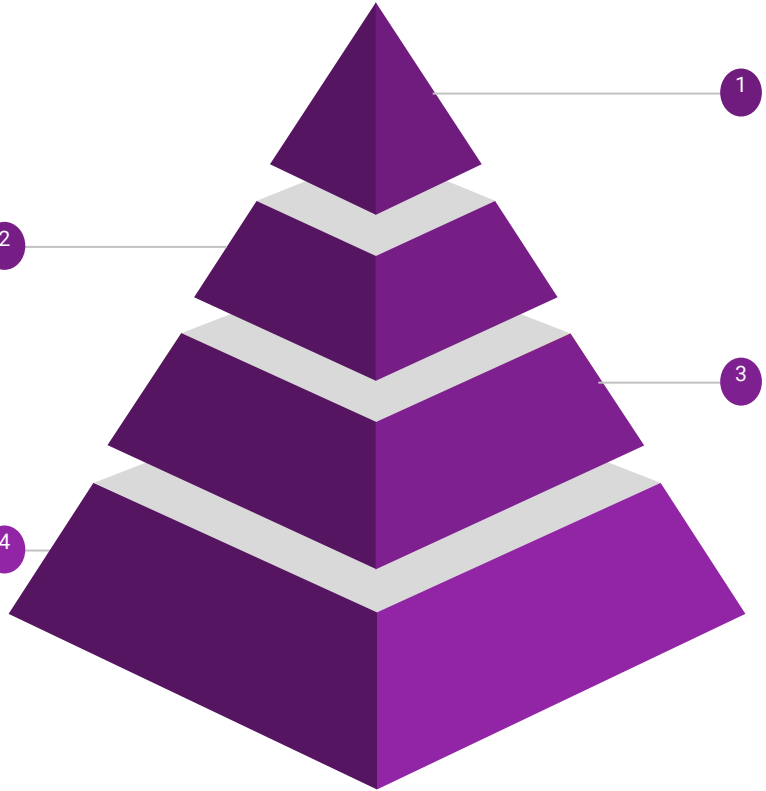
Why is SLAM a Hard Problem?

1. Robot path and map are both **unknown**



2. Map and pose estimates correlated

Reference : https://www.mrpt.org/tutorials/slam-algorithms/slam-simultaneous-localization-and-mapping-for-beginners-the-basics/



1 **Hardware Implementation**

Coordination

2 Swarm Intelligence, Flocking, Foraging

Control

3 Path planning, PID, RL, A*.

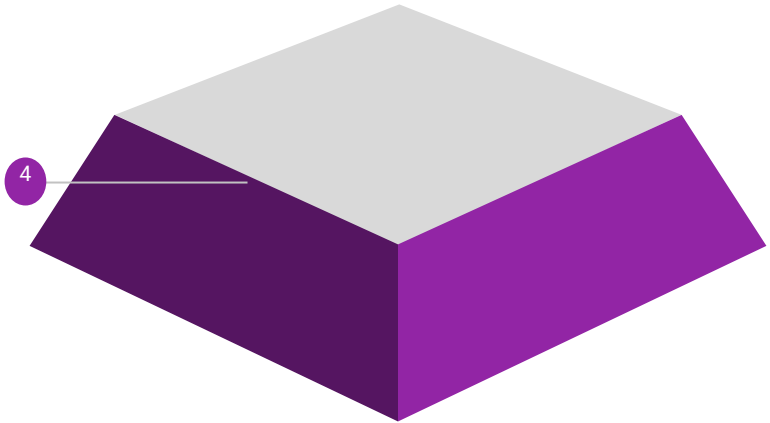
Perception

4 Slam algorithms - Gmapping, RTABMAP, Hector, Karto. (3D cameras, Lidars).

SLAM

Perception ??

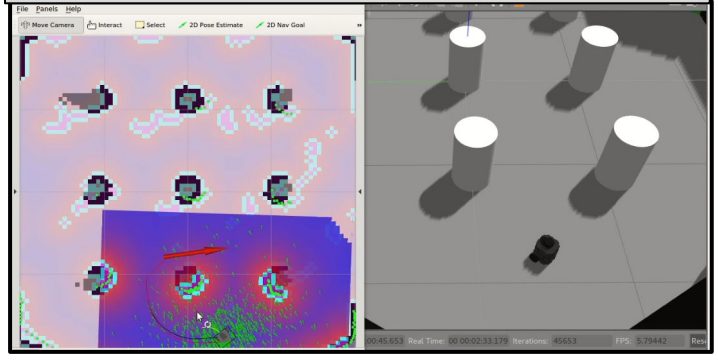
Slam algorithms - Gmapping, RTABMAP, Hector, Karto. (3D cameras, Lidars).



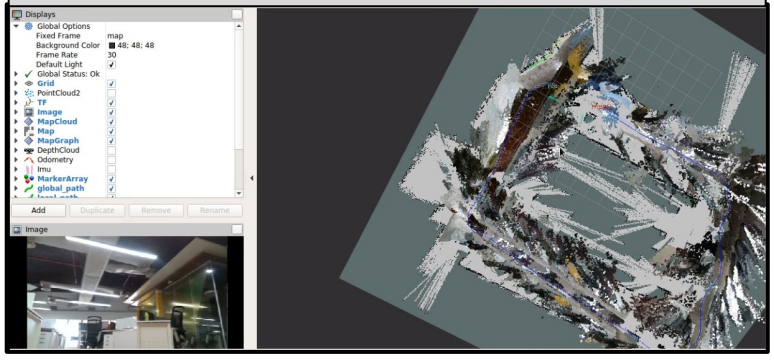
SLAM

PERCEPTION

GMAPPING

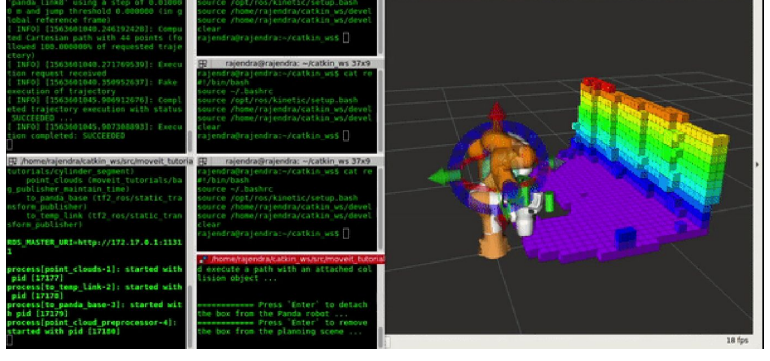


RTABMAP

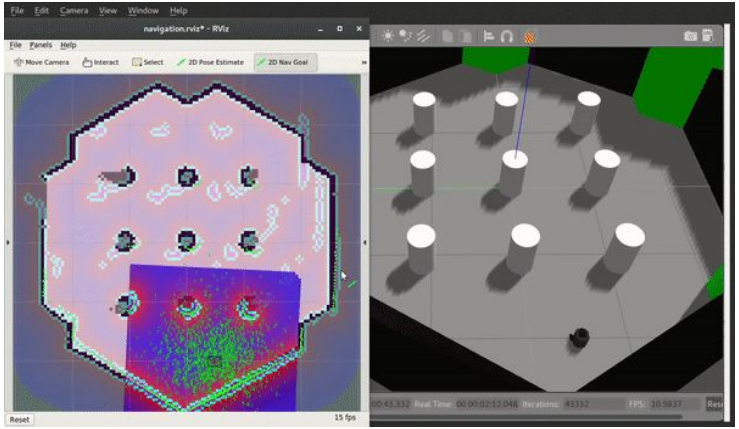


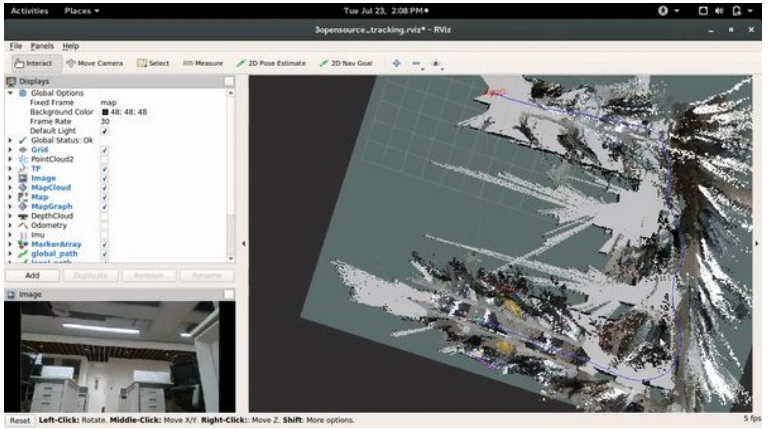
Point cloud processing

Point cloud Reconstruction

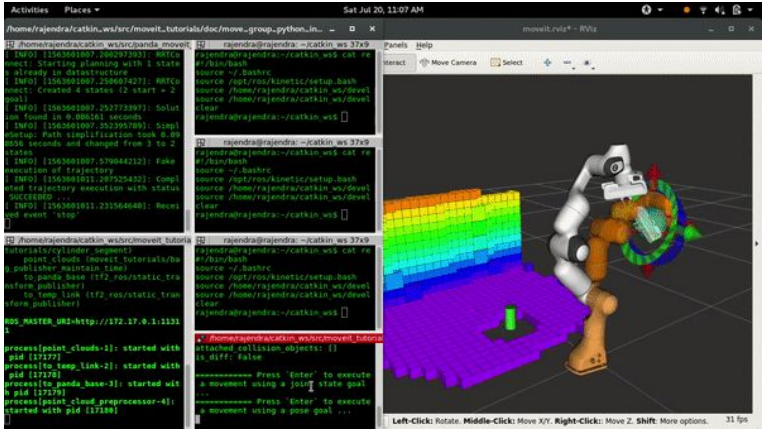


PERCEPTION


File Edit Camera View Window Help
navigation.rviz* - RViz
File Panels Help
Move Camera
Interact
Select
2D Pose Estimate
2D Nav Goal
Reset
15 fps


Activities
Places
Tue Jul 23, 2:08 PM
3opensource_tracking.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Measure
2D Pose Estimate
2D Nav Goal
Displays
Global Options
Fixed Frame
map
Background Color
48; 48; 48
Frame Rate
30
Default Light
Global Status: Ok
Grid
PointCloud2
TF
Image
MapCloud
Map
MapGraph
DepthCloud
Odometry
Imu
MarkerArray
global_path
Add
Duplicate
Remove
Rename
Image
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Move Z. Shift: More options.
5 fps

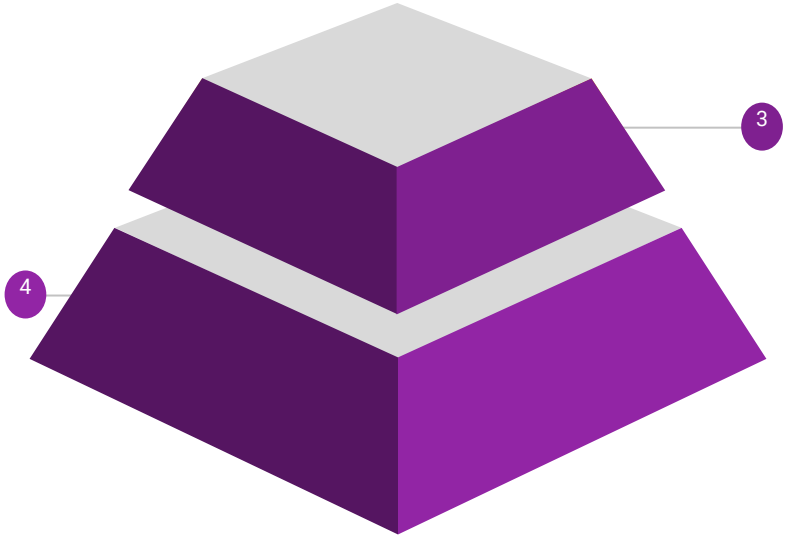

Activities
Places
Mon Jul 22, 4:46 PM
PCD viewer


Activities
Places
Sat Jul 20, 11:07 AM
moveit.rviz* - RViz
Panels Help
Interact
Move Camera
Select
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Move Z. Shift: More options.
31 fps

Control ??

3

Path planning, PID, RL, A*.



Perception ✓✓

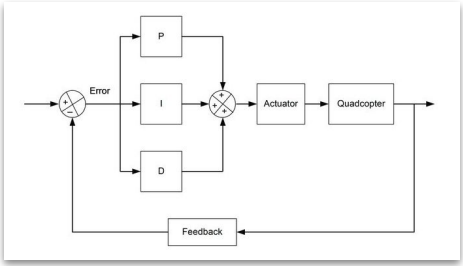
4

Slam algorithms - Gmapping, RTABMAP, Hector, Karto. (3D cameras, Lidars).

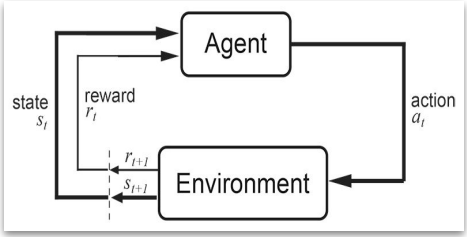
SLAM

CONTROL

Point A
(Start)

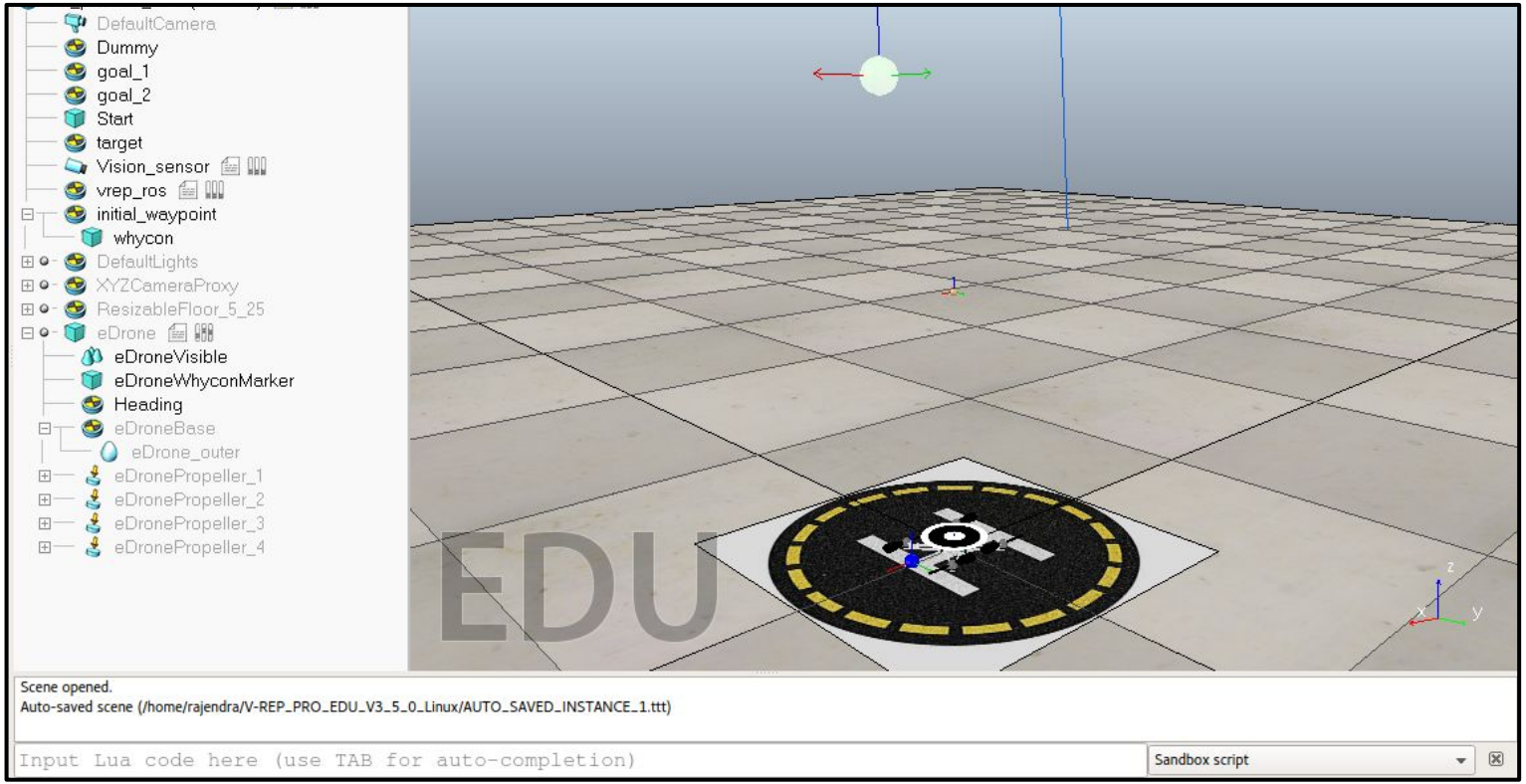


Point B
(End)

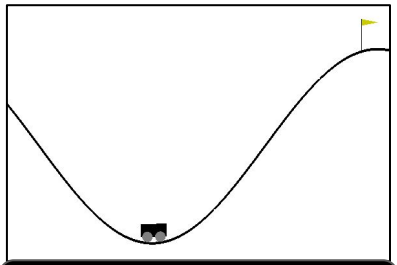


VREP SIMULATION

Pluto Drone

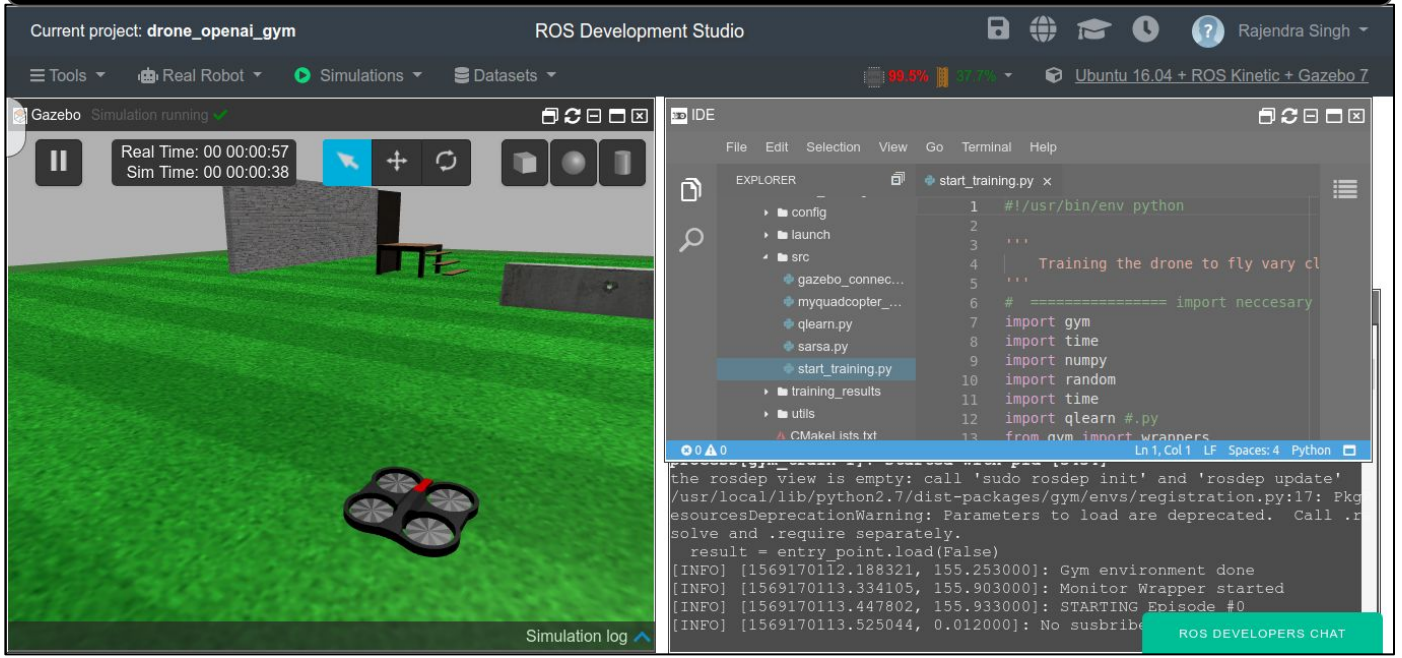


ARdrone



Mountain car

ROS DEVELOPMENT STUDIO



Q-learning

Q[s,a] = lr * Q[s,a] + (1 - lr) * { currentReward + discountFactor * max(Q[s',:])}

s : given state **, a :** action(ε-greedy)**, s' -** next state, **lr** - learning rate

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state},\cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$
  until S is terminal

Ref - https://towardsdatascience.com/reinforcement-learning-temporal-difference-sarsa-q-learning-expected-sarsa-on-python-9fecfda7467e

SARSA

Sarsa

Q[s,a] = lr * Q[s,a] + (1 - lr) * { currentReward + discountFactor * Q[s',a'] }

s : given state **, a :** action(ε-greedy), a' - next action**, s' -** next state, **lr** - learning rate

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Ref - https://towardsdatascience.com/reinforcement-learning-temporal-difference-sarsa-q-learning-expected-sarsa-on-python-9fecfda7467e

File Edit Selection View Go Debug Terminal Help

EXPLORER

OPEN EDITORS
- sarsa.py ~/Slam_and_RL_BTP/code/AI/drone
- moutaincar_qlearn.py AI_lab • gym/mountain... M
- moutaincar_sarsa.py AI_lab • gym/mountainC... M
- 1.py AI_lab • gym/mountainCar/sarsa U
- main.py ~/Slam_and_RL_BTP/code/swarm/drone U

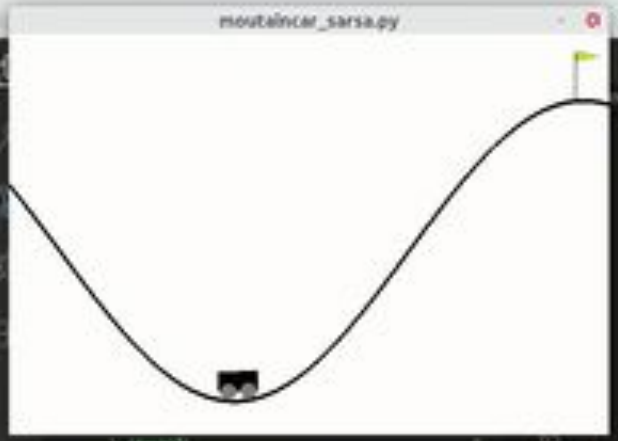
UNTITLED (WORKSPACE)
- gym
  - keras-rl
  - mountainCar
    - dqn
    - qlearn
      - rewards
      - moutaincar_qlearn.py M
    - sarsa
      - rewards
      - 1.py U
      - moutaincar_sarsa.py M
  - moutaincar_randomwalk.py
  - moutaincar_sarsa.py M
  - moutaincar.py
- acrobot.py
- cartpole_randomwalk.py
- frozenLake_sarsa.py
- moutaincar_qlearn.py
- rewards_qlearn.jpg
- rewards.jpg
- labtest2
- week1
- week2
- README.md

OUTLINE

sarsa.py | moutaincar_qlearn.py | moutaincar_sarsa.py | 1.py | main.py

AI_lab > gym > mountainCar > sarsa > moutaincar_sarsa.py > Sarsa

```
52      epd = (ep-minep)/e# Calculate episodic epd in ep
53      #========== for e episodes ============#
54      for i in range(e):
55          done = False
56          rsum,R = 0,0
57          s = env.reset() #return (x,y)
58          sd = discret(s)# Discretize
59          #========== while not done ============#
60          if i == (e - 20):
61                  print("\n********** Press Enter run ******** \n")
62                  input()
63          while done != True:
64              if i >= (e - 20):# Render environment for last five e
65                  env.render()
66                  sleep(0.01)
67              #====== explore vs exploit =====#
68              if np.random.random() < 1 - ep:#epsilon greedy exr vs expt
69                  a = np.argmax(Q[sd[0], sd[1]])
70              else:
71                  a = np.random.randint(0, env.action_space.n)
72              s_, R, done, info = env.step(a)
73              s_d = discret(s_)# Discretize
74              #====== update q =====#
75              if done and s_[0] >= 0.5:#Allow for terminal states
76                  Q[sd[0], sd[1], a] = R
77              else:# Adjust Q value for current s
78
79                  if np.random.random() < 1 - ep:#epsilon greedy exr vs expt
80                      a_ = np.argmax(Q[s_d[0], s_d[1]])
81                  else:
82                      a_ = np.random.randint(0, env.action_space.n)
83
84                  # Q[sd[0], sd[1],a] =(1-lr)*Q[sd[0],sd[1],a]  +   lr*(R + gamma*np.max(Q[s_d[0], s_d[1]]))
85                  Q[sd[0], sd[1],a] =(1-lr)*Q[sd[0],sd[1],a]  +   lr*(R + gamma*Q[s_d[0], s_d[1],a_])
86              #====== variable update =====#
87              rsum += R# Update variables
88              sd = s_d
89          #====== epsilon decay =====#
```

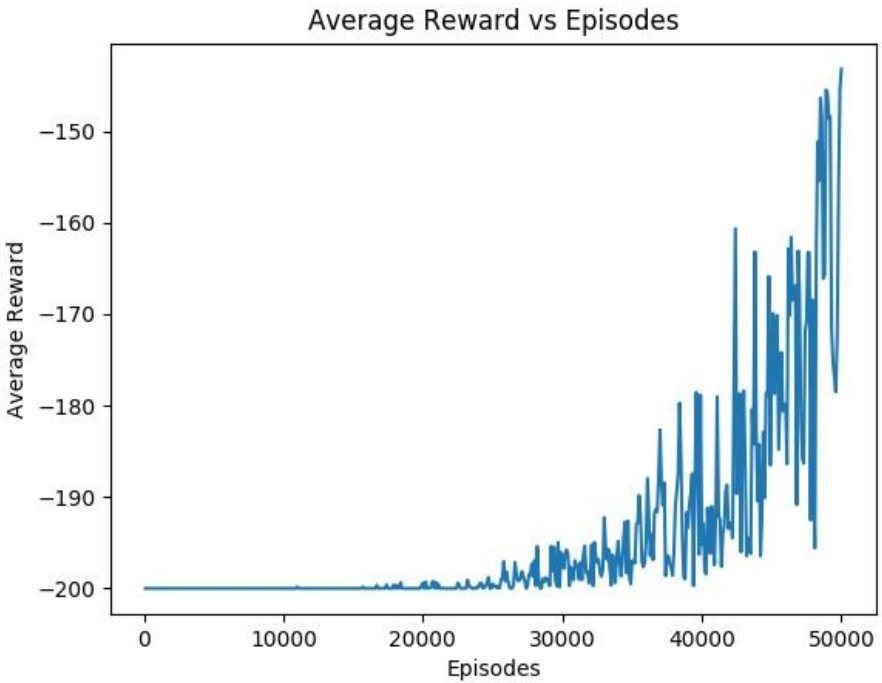



```
for i in range(e):
    done = False
    rsum,R = 0,0
    s = env.reset() #return (x,y)
    sd = discret(s)# Discretize
    ########## while not done ##########
    if i == (e - 20):
            print("\n********** Press Enter run ******** \n")
            input()
    while done != True:
        if i >= (e - 20):# Render environment for last five e
            env.render()
            sleep(0.01)
        ###### explore vs exploit #####
        if np.random.random() < 1 - ep:#epsilon greedy exr vs expt
            a = np.argmax(Q[sd[0], sd[1]])
        else:
            a = np.random.randint(0, env.action_space.n)
        s_, R, done, info = env.step(a)
        s_d = discret(s_)# Discretize
        ###### update q #####
        if done and s_[0] >= 0.5:#Allow for terminal states
            Q[sd[0], sd[1], a] = R
        else:# Adjust Q value for current s
```

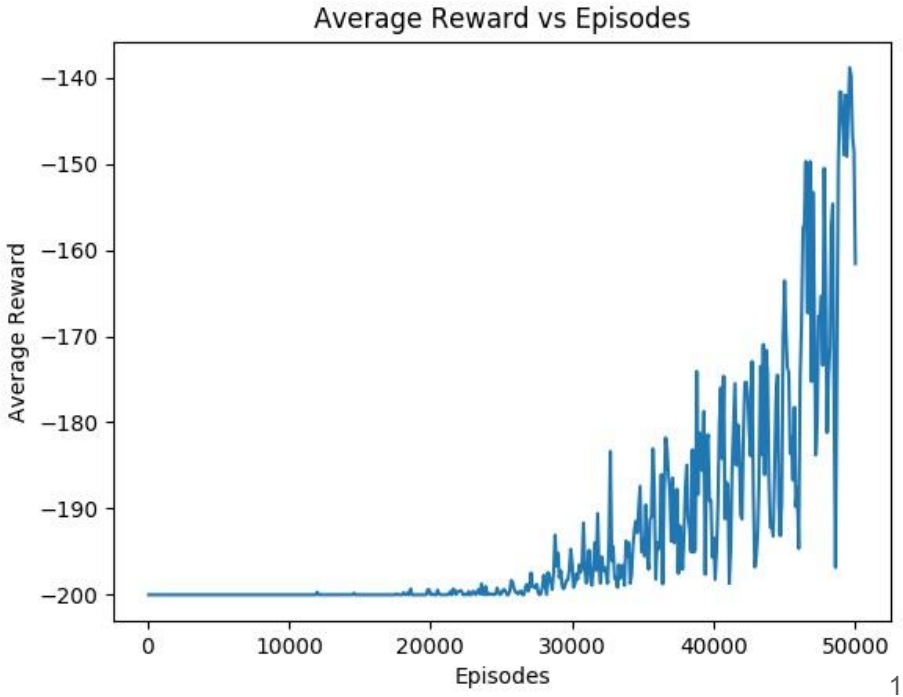
```
Episode 49500 Average Reward: -146.01
Episode 49600 Average Reward: -153.11
Episode 49700 Average Reward: -164.5
Episode 49800 Average Reward: -165.06
Episode 49900 Average Reward: -171.38

********** Press Enter run ********
```

```
Episode 49600 Average Reward: -170.84
Episode 49700 Average Reward: -149.2
Episode 49800 Average Reward: -155.84
Episode 49900 Average Reward: -162.52

********** Press Enter run ********
```

Q-learning



SARSA



SARSA ON DRONE

TRAINING

```
# for each episode, we test the robot for nsteps
for i in range(nsteps):

    # Pick an action based on the current state
    action = qlearn.chooseAction(state)

    # Execute the action in the environment and get feedback
    observation, reward, done, info = env.step(action)
    cumulated_reward += reward
    if highest_reward < cumulated_reward:
        highest_reward = cumulated_reward

    nextState = ''.join(map(str, observation))

    # Make the algorithm learn based on the results
    qlearn.learn(state, action, reward, nextState)

    if not(done):
        state = nextState
    else:
        rospy.loginfo ("DONE")
        last_time_steps = numpy.append(last_time_steps, [int(i + 1)])
        break
```

SARSA CHOOSE ACTION

```
def chooseAction(self, state):
    if random.random() < self.epsilon:
        action = random.choice(self.actions)
    else:
        q = [self.getQ(state, a) for a in self.actions]
        maxQ = max(q)
        count = q.count(maxQ)
        if count > 1:
            best = [i for i in range(len(self.actions)) if q[i] == maxQ]
            i = random.choice(best)
        else:
            i = q.index(maxQ)

        action = self.actions[i]
    return action

def learn(self, state1, action1, reward, state2, action2):
    qnext = self.getQ(state2, action2)
    self.learnQ(state1, action1, reward, reward + self.gamma * qnext)
```

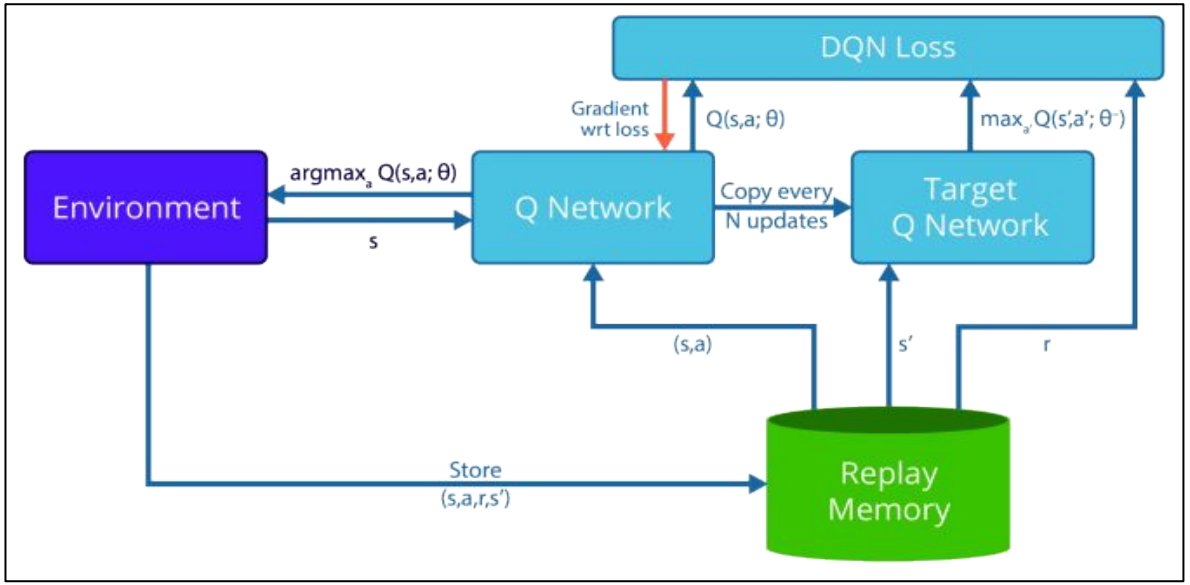
GAZEBO

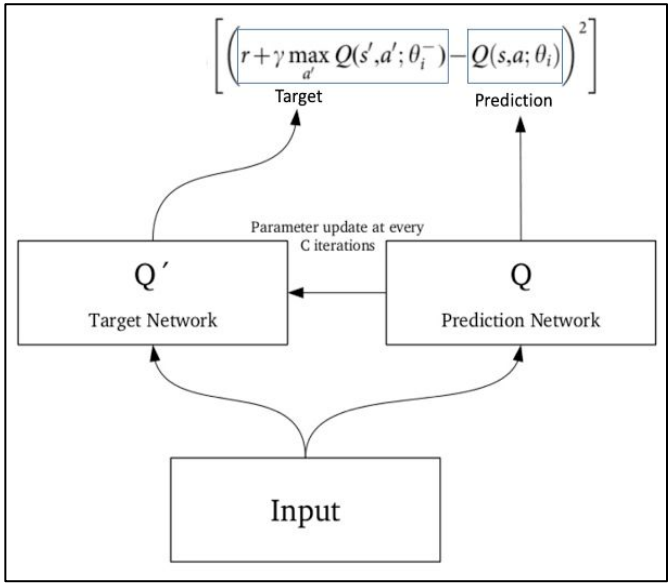
```
def pauseSim(self):
    rospy.wait_for_service('/gazebo/pause_physics')
    try:
        self.pause()
    except rospy.ServiceException, e:
        print ("/gazebo/pause_physics service call failed")
def unpauseSim(self):
    rospy.wait_for_service('/gazebo/unpause_physics')
    try:
        self.unpause()
    except rospy.ServiceException, e:
        print ("/gazebo/unpause_physics service call failed")
def resetSim(self):
    rospy.wait_for_service('/gazebo/reset_simulation')
    try:
        self.reset_proxy()
    except rospy.ServiceException, e:
        print ("/gazebo/reset_simulation service call failed")
```

OPEN GYM ENVIRONMENT

```
def take_observation (self):
    data_pose = None
    while data_pose is None:
        try:
            data_pose = rospy.wait_for_message('/drone/gt_pose', Pose, timeout=5)
        except:
            rospy.loginfo("Current drone pose not ready yet, retrying for getting robot pose")
    data_imu = None      You, 2 days ago • updated
    while data_imu is None:
        try:
            data_imu = rospy.wait_for_message('/drone/imu', Imu, timeout=5)
        except:
            rospy.loginfo("Current drone imu not ready yet, retrying for getting robot imu")
    return data_pose, data_imu
def calculate_dist_between_two_Points(self,p_init,p_end):
    a = np.array((p_init.x ,p_init.y, p_init.z))
    b = np.array((p_end.x ,p_end.y, p_end.z))
    dist = np.linalg.norm(a-b)
    return dist
def init_desired_pose(self):
```

Deep Q-Network (DQN)

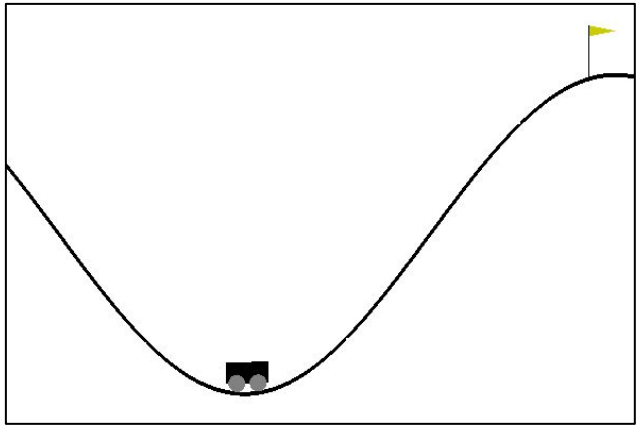




Ref : https://datascience.stackexchange.com/questions/37643/how-to-use-a-different-model-to-deep-neural-network-with-reinforcement-learning

Starting State

Random position from -0.6 to -0.4 with no velocity.



Episode Termination

The episode ends when you reach 0.5 position, or if 200 iterations are reached.

Reward

-1 for each time step, until the goal position of 0.5 is reached.

Observation

Type: Box(2)

Num	Observation	Min	Max
0	position	-1.2	0.6
1	velocity	-0.07	0.07

Actions

Type: Discrete(3)

Num	Action
0	push left
1	no push
2	push right

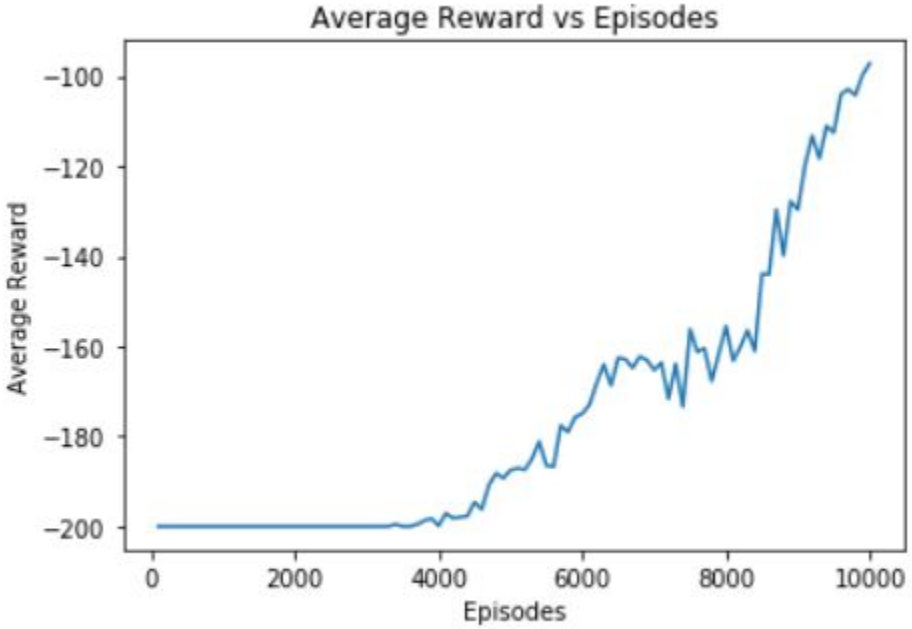
Reference - https://github.com/openai/gym/wiki/MountainCar-v0

Chosen parameter

```
gamma=0.99
learingRate=0.001
episodeNum=10000
stepNum=201 #max is 200 #steps in one episode

epsilon = 1
epsilon_min=0.01
epsilon_decay = 0.01
epsilon_decay = (epsilon-epsilon_min)/episodeNum
print(epsilon_decay)
targetNetworkUpdateRate=10

replayBuffer=deque(maxlen=20000)
numPickFromBuffer=32
```

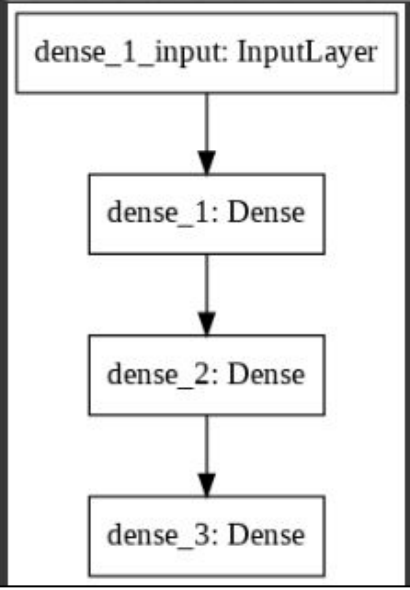
```
def learn():
    global gamma,learingRate,stepNum,epsilon,epsilon_decay,epsilon_min,numPickFromBuffer
    if len(replayBuffer) < numPickFromBuffer:
        return
    samples = random.sample(replayBuffer,numPickFromBuffer)
    states,newStates = [],[]
    for sample in samples:
        state, action, reward, new_state, done = sample
        states.append(state)
        newStates.append(new_state)
    states = np.array(states).reshape(numPickFromBuffer, 2)
    newStates = np.array(newStates).reshape(numPickFromBuffer, 2)
    #[  [Q(s1,a1), Q(s1,a2), Q(s1,a3)],
    #   [Q(s2,a1), Q(s2,a2), Q(s2,a3)],
    #   [Q(s3,a1), Q(s3,a2), Q(s3,a3)]  ]
    y_Qs = trainNetwork.predict(states)
    y_Qs_dash=targetNetwork.predict(newStates) #similarly Q[s',a1..a2...] for states
    y_Qs_star = copy.deepcopy(y_Qs) # initialise as of now, to be calculate later

    for j,sample in enumerate(samples):
        state, action, reward, new_state, done = sample
        if done:
            y_Qs_star[j][action] = reward
        else:
            y_Qs_star[j][action] = reward + gamma * max(y_Qs_dash[j])

    return trainNetwork.fit(states, y_Qs_star, epochs=1, verbose=0)
```

```
def createNetwork():
    global gamma,learingRate,stepNum,epsilon,epsilon_decay,epsilon_min,numPickFromBuffer
    model = models.Sequential()
    state_shape = env.observation_space.shape
    model.add(layers.Dense(24, activation='relu', input_shape=state_shape))
    model.add(layers.Dense(48, activation='relu'))
    model.add(layers.Dense(env.action_space.n,activation='linear'))
    model.compile(loss='mse', optimizer=Adam(lr=learingRate))
    return model
```

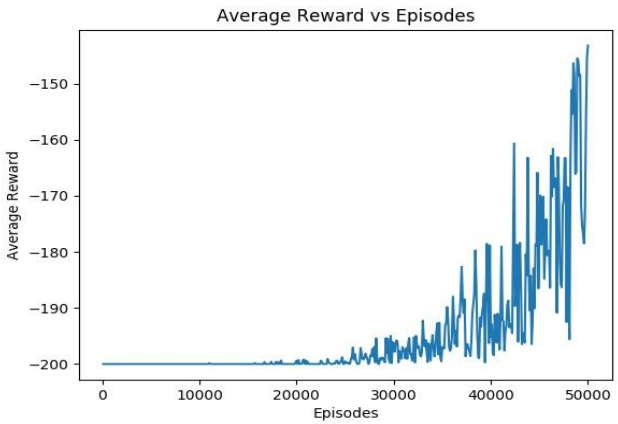



```
episode : 8700 ,    epsilon is 0.14,      reward is -129,      maxPosition is 0.50
episode : 8800 ,    epsilon is 0.13,      reward is -139,      maxPosition is 0.52
episode : 8900 ,    epsilon is 0.12,      reward is -127,      maxPosition is 0.50
episode : 9000 ,    epsilon is 0.11,      reward is -129,      maxPosition is 0.51
episode : 9100 ,    epsilon is 0.10,      reward is -119,      maxPosition is 0.51
episode : 9200 ,    epsilon is 0.09,      reward is -113,      maxPosition is 0.50
episode : 9300 ,    epsilon is 0.08,      reward is -118,      maxPosition is 0.51
episode : 9400 ,    epsilon is 0.07,      reward is -111,      maxPosition is 0.52
episode : 9500 ,    epsilon is 0.06,      reward is -112,      maxPosition is 0.51
episode : 9600 ,    epsilon is 0.05,      reward is -104,      maxPosition is 0.50
episode : 9700 ,    epsilon is 0.04,      reward is -102,      maxPosition is 0.53
episode : 9800 ,    epsilon is 0.03,      reward is -104,      maxPosition is 0.51
episode : 9900 ,    epsilon is 0.02,      reward is -99,      maxPosition is 0.51
episode : 10000 ,    epsilon is 0.01,      reward is -97,      maxPosition is 0.52
```

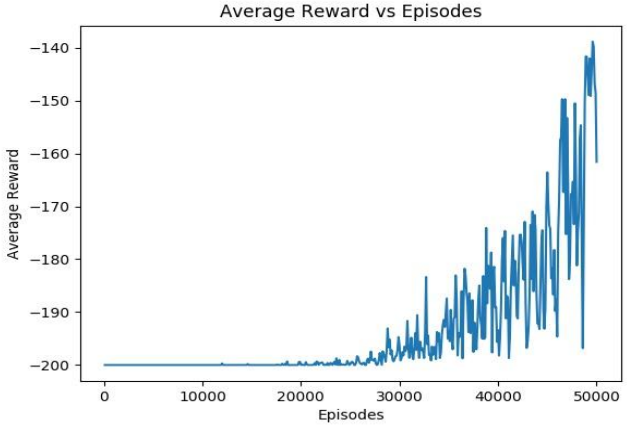
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 24)                72
_________________________________________________________________
dense_2 (Dense)              (None, 48)                1200
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 147
=================================================================
Total params: 1,419
Trainable params: 1,419
Non-trainable params: 0
_________________________________________________________________
```



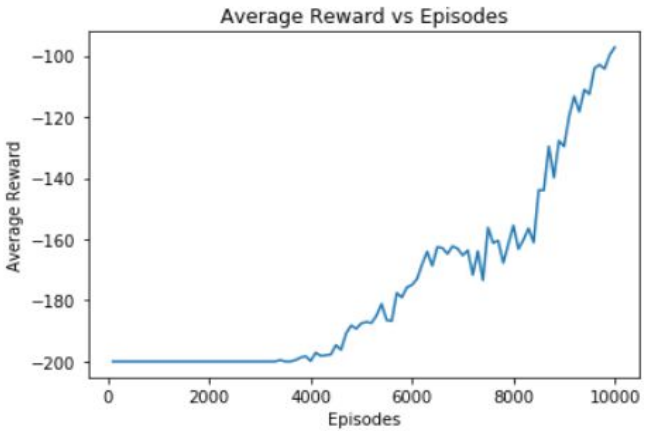
Q-learning



SARSA

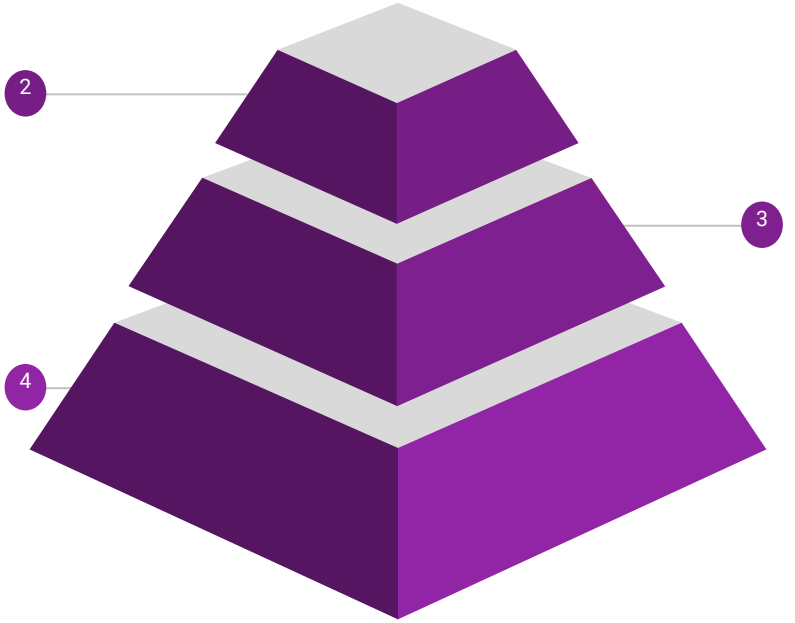


DQN



Coordination ??

Swarm Intelligence, Flocking, Foraging



Control ✓✓

Path planning, PID, RL, A*.

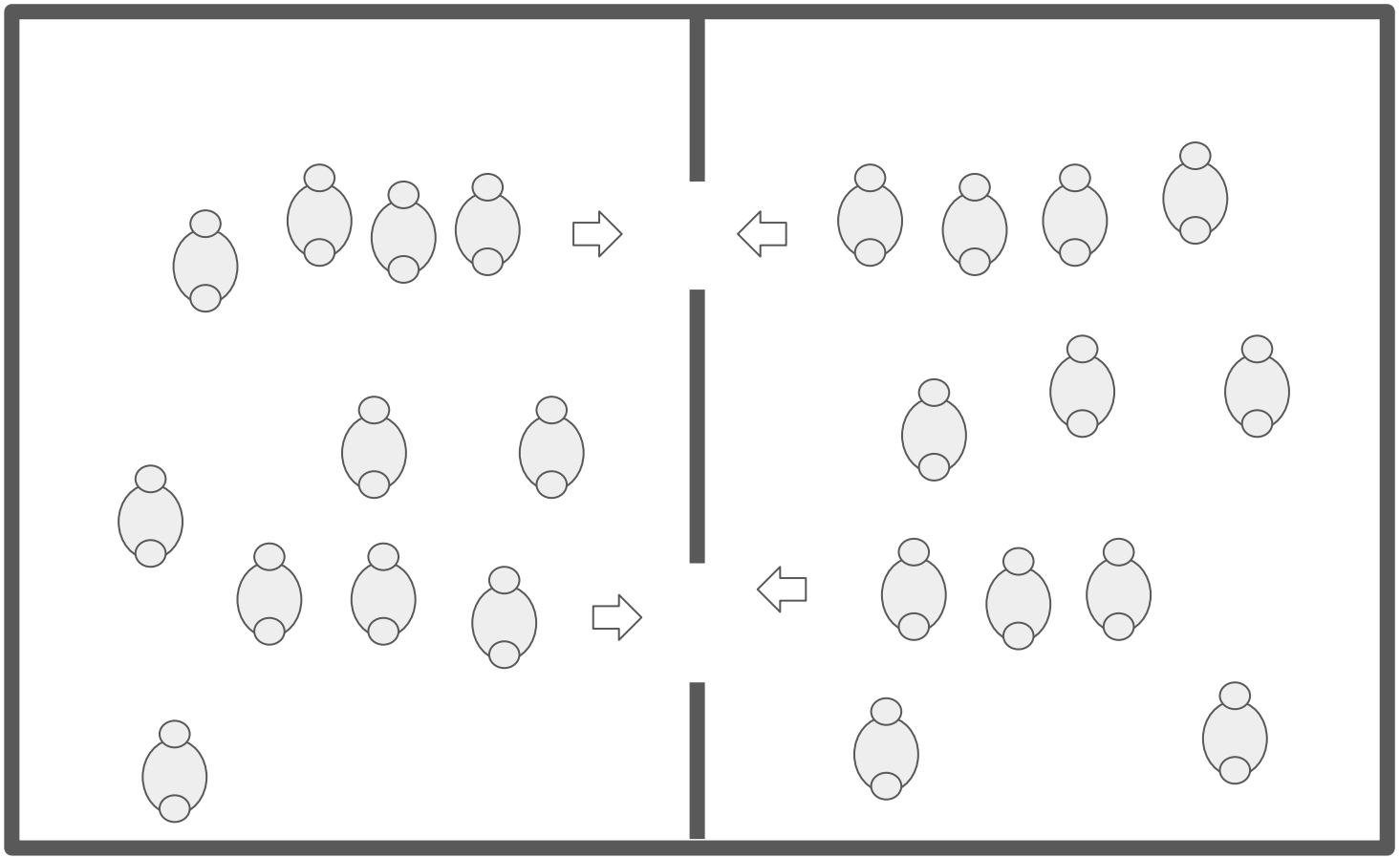
Perception ✓✓

Slam algorithms - Gmapping, RTABMAP, Hector, Karto. (3D cameras, Lidars).

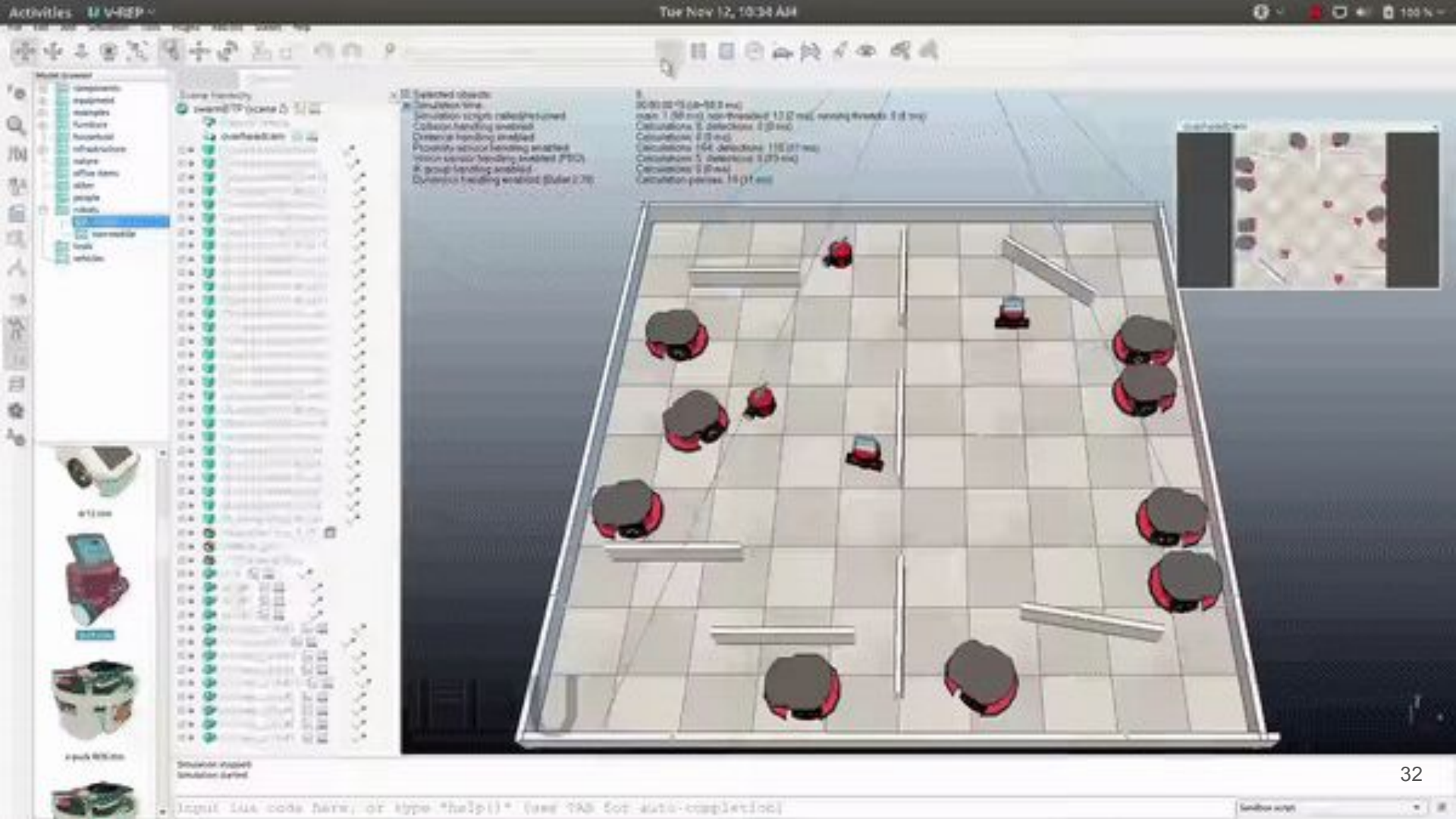
SLAM

Claim 1. We need good coordination in robot to perform any real life task !

Experiment 1.



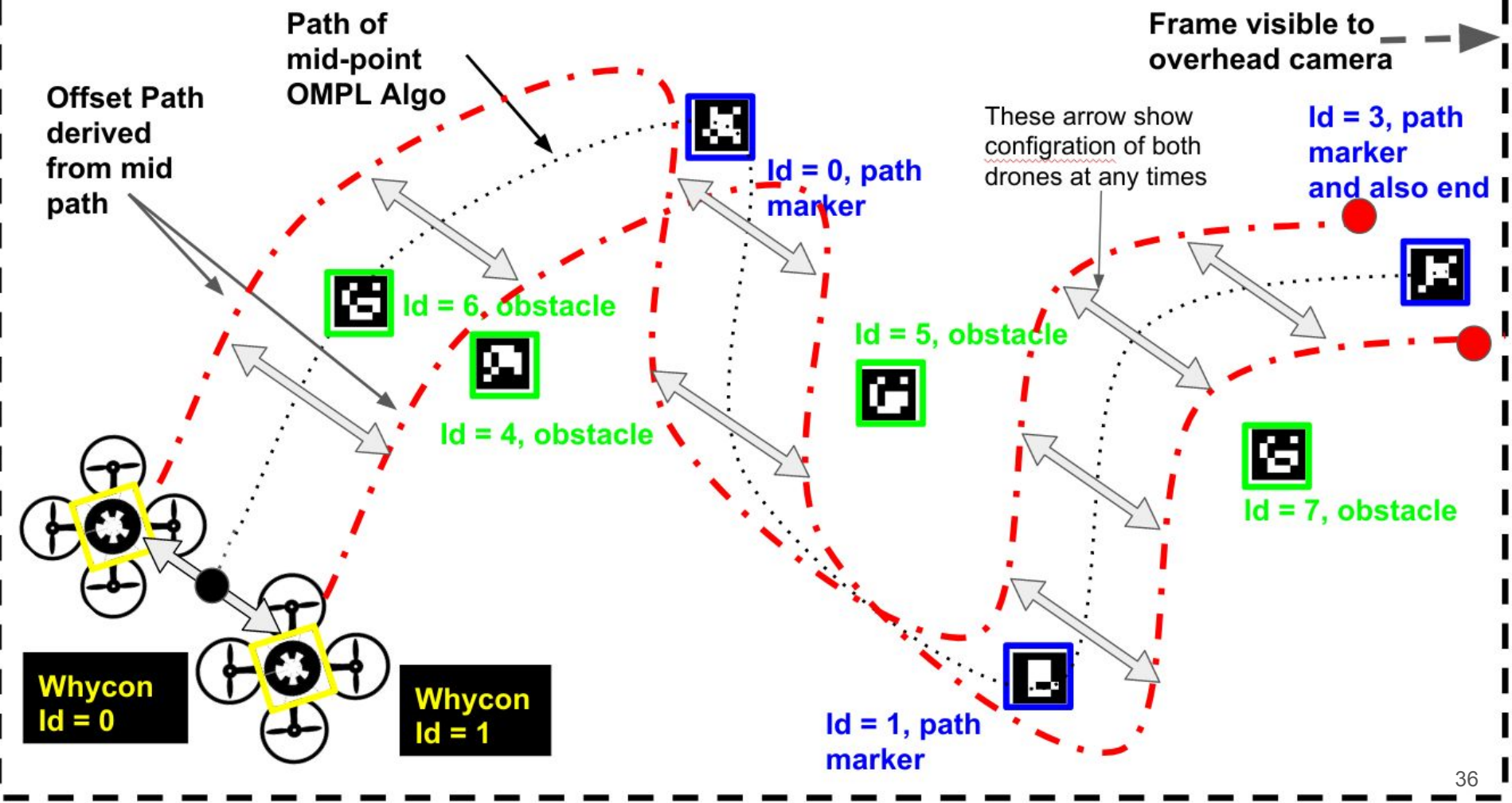
V-REP
File Edit Add Simulation Tools Plugins Add-ons Scenes Help
Model browser
components
equipment
examples
furniture
household
infrastructure
nature
office items
other
people
robots
mobile
non-mobile
tools
vehicles
new scene
swarmBTP
Scene hierarchy
swarmBTP (scene 2)
DefaultCamera
overheadcam
20cmHighWall100cm
20cmHighWall100cm1
20cmHighWall100cm10
20cmHighWall100cm11
20cmHighWall100cm12
20cmHighWall100cm13
20cmHighWall100cm14
20cmHighWall100cm15
20cmHighWall100cm16
20cmHighWall100cm17
20cmHighWall100cm18
20cmHighWall100cm19
20cmHighWall100cm20
20cmHighWall100cm21
20cmHighWall100cm22
20cmHighWall100cm23
20cmHighWall100cm24
20cmHighWall100cm25
20cmHighWall100cm26
20cmHighWall100cm27
20cmHighWall100cm29
20cmHighWall100cm3
20cmHighWall100cm4
20cmHighWall100cm5
20cmHighWall100cm6
20cmHighWall100cm7
20cmHighWall100cm8
20cmHighWall100cm9
ResizableFloor_5_25
DefaultLights
XYZCameraProxy
dr20
dr20#0
dr20#1
dr20#2
Pioneer_p3dx#7
Pioneer_p3dx
Pioneer_p3dx#0
Pioneer_p3dx#1
Pioneer_p3dx#10
Pioneer_p3dx#2
Pioneer_p3dx#5
Pioneer_p3dx#8
Pioneer_p3dx#9
Selected objects: 0
Simulation time: 00:00:00.90 (dt=50.0 ms)
overheadcam
7 DoF manipulator.ttm
ABB IRB 140.ttm
ABB IRB 360.ttm
Simulation started.
Simulation suspended.
31
Input Lua code here, or type "help()" (use TAB for auto-completion)
Sandbox script


Activities
V-REP
Tue Nov 12, 10:34 AM

01	**No communication**	• Hence each member need to think for itself
02	**Become hurdle for other**	• Deadlock situation
03	**No fast convergence**	• Even if it work then also it will take time

Claim 2. When I say coordination I doesn’t really mean centralized !

Experiment 2.


Path of mid-point OMPL Algo
Offset Path derived from mid path
Frame visible to overhead camera
These arrow show configration of both drones at any times
Id = 0, path marker
Id = 3, path marker and also end
Id = 6, obstacle
Id = 5, obstacle
Id = 4, obstacle
Id = 7, obstacle
Id = 1, path marker
Whycon Id = 0
Whycon Id = 1

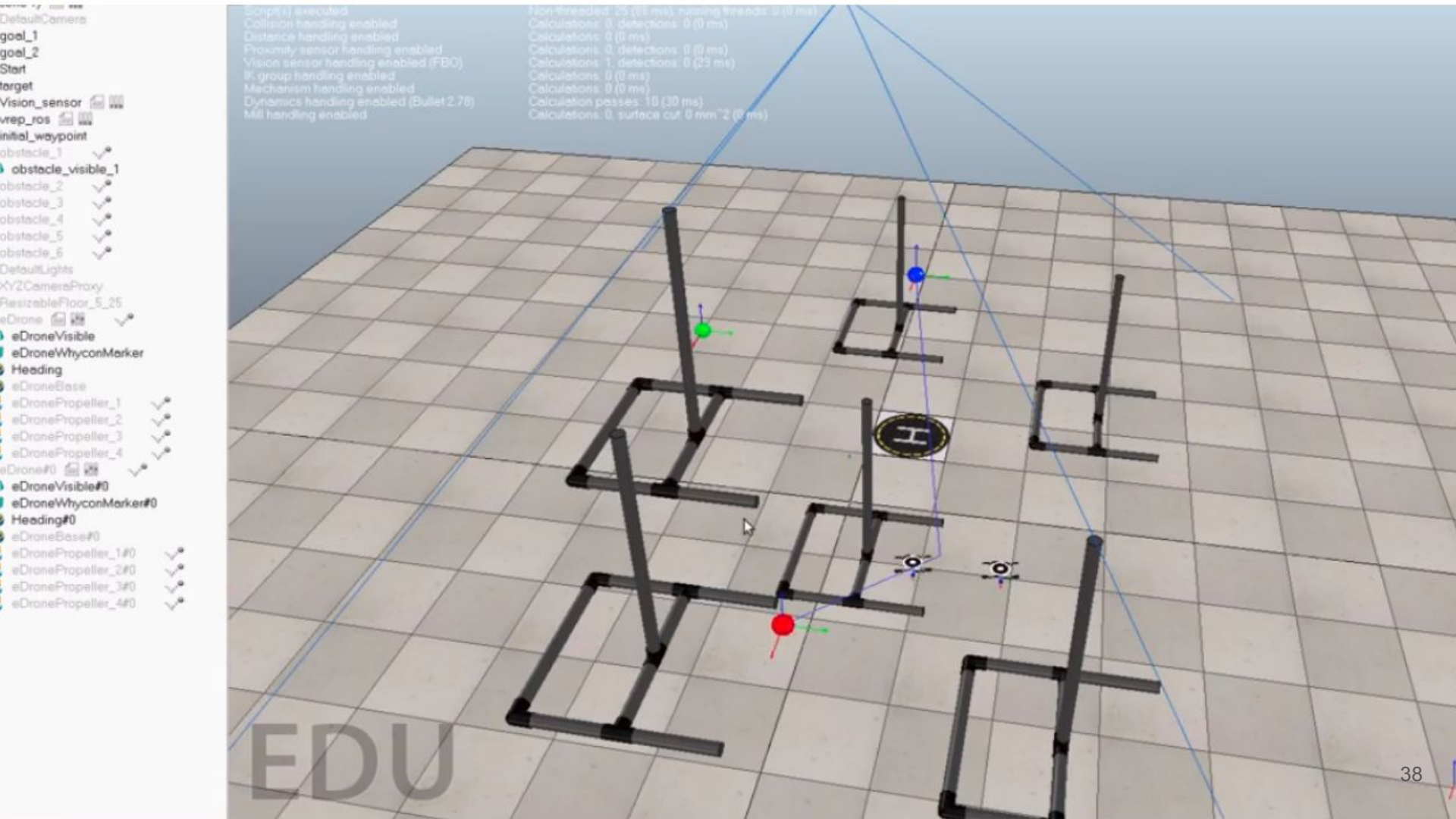
Code

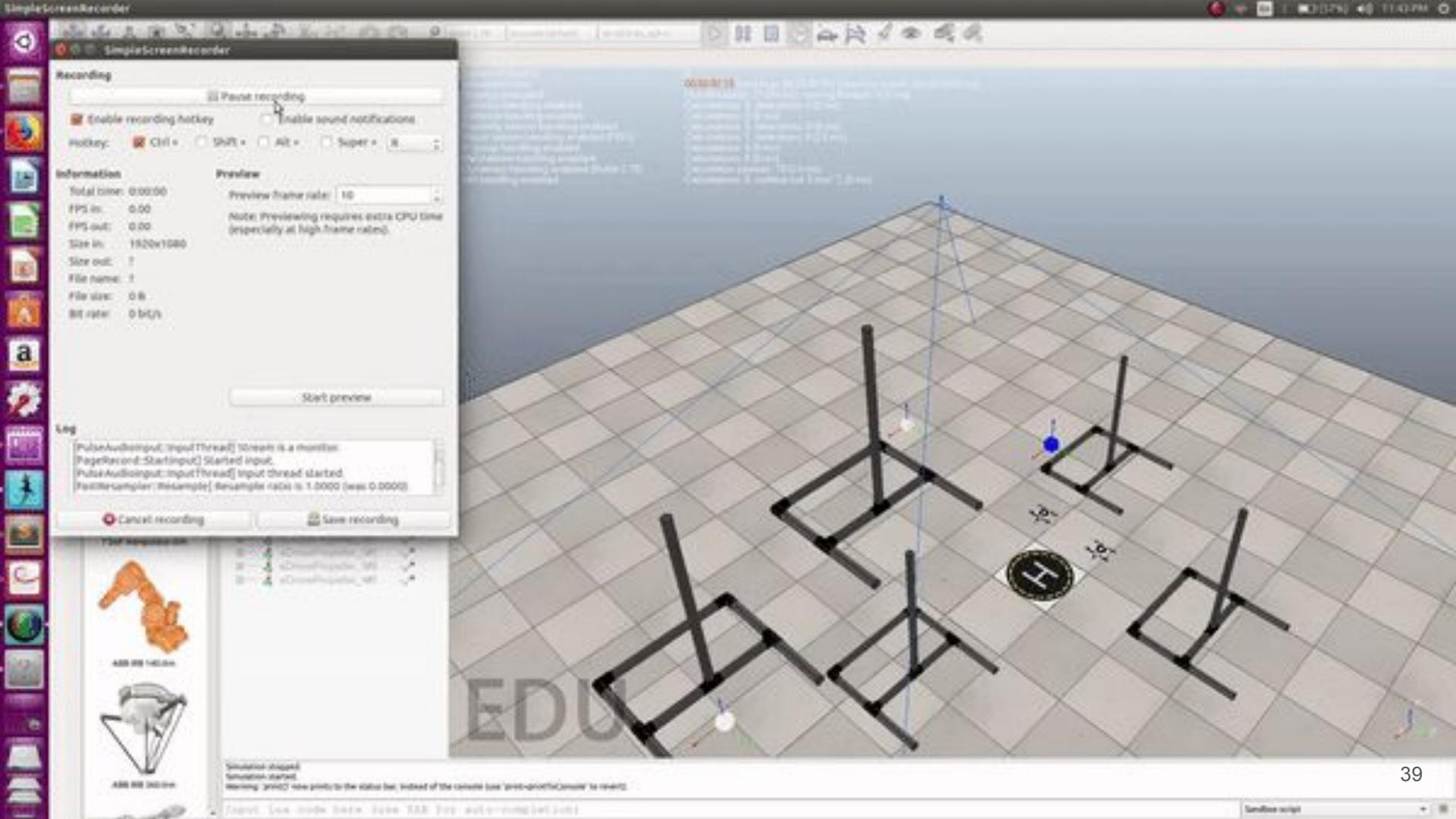
sarsa.py | moutaincar_qlearn.py | moutaincar_sarsa.py | 1.py | main.py

home > rajendra > Slam_and_RL_BTP > code > swarm > drone > main.py

```python
if __name__ == '__main__':
    pid_class = PID()
    error = []
    # Go to intial set point
    while True:
        error = pid_class.pid(pid_class.initialSetPoint)
        if (abs(error[0]) <= 0.5 and abs(error[1]) <= 0.5 and abs(error[2]) <= 0.5):
            break
    #go to one by one checkpoint
    for i in range (0, 3):
        if i != 0:
            print("next_target")
        pid_class.next.publish('1') #signal to sent next check point
        rospy.sleep(0.5)
        length = len(pid_class.setPoint.poses)
        for j in range (0, length): #go to one by one subsetpoint
            temp = [0.0, 0.0, 0.0, 0.0]
            temp[0] = pid_class.setPoint.poses[j].position.x
            temp[1] = pid_class.setPoint.poses[j].position.y
            temp[2] = pid_class.setPoint.poses[j].position.z
            temp[3] = 0.0
            while (True): # while constrained not meet
                error = pid_class.pid(temp)
                if (error[0] <= 0.5 and error[1] <= 0.5 and error[2] <= 0.5):
                    break
    #land at start point
    while True:
        error = pid_class.pid(pid_class.start) #find error start point
        # if constrained meet then disarm (i.e disarm near ground)
        if (abs(error[0]) <= 0.5 and abs(error[1]) <= 0.5 and abs(error[2]) <= 1):
            pid_class.disarm()
            break
```

Reference : https://medium.com/better-programming/boids-simulating-birds-flock-behavior-in-python-9fff99375118


DefaultCamera
goal_1
goal_2
Start
target
Vision_sensor
vrep_ros
initial_waypoint
obstacle_1
obstacle_visible_1
obstacle_2
obstacle_3
obstacle_4
obstacle_5
obstacle_6
DefaultLights
XYZCameraProxy
ResizableFloor_5_25
eDrone
eDroneVisible
eDroneWhyconMarker
Heading
eDroneBase
eDronePropeller_1
eDronePropeller_2
eDronePropeller_3
eDronePropeller_4
eDrone#0
eDroneVisible#0
eDroneWhyconMarker#0
Heading#0
eDroneBase#0
eDronePropeller_1#0
eDronePropeller_2#0
eDronePropeller_3#0
eDronePropeller_4#0
Collision handling enabled
Distance handling enabled
Proximity sensor handling enabled
Vision sensor handling enabled (FBO)
IK group handling enabled
Mechanism handling enabled
Dynamics handling enabled (Bullet 2.78)
Mill handling enabled
Calculation passes: 10 (30 ms)
EDU

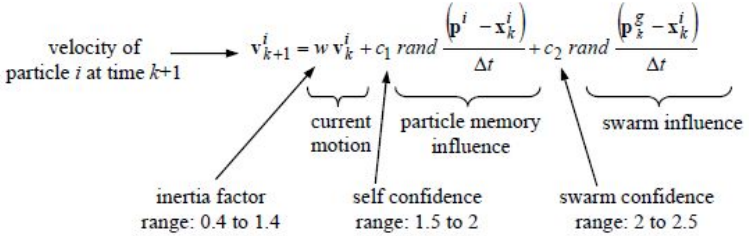

SimpleScreenRecorder
Recording
Pause recording
Enable recording hotkey
Enable sound notifications
Hotkey:
Ctrl +
Shift +
Alt +
Super +
R
Information
Total time: 0:00:00
FPS in: 0.00
FPS out: 0.00
Size in: 1920x1080
Size out: ?
File name: ?
File size: 0 B
Bit rate: 0 bit/s
Preview
Preview frame rate: 10
Note: Previewing requires extra CPU time (especially at high frame rates).
Start preview
Log
[PulseAudioInput::InputThread] Stream is a monitor.
[PageRecord::StartInput] Started input.
[PulseAudioInput::InputThread] Input thread started.
[FastResampler::Resample] Resample ratio is 1.0000 (was 0.0000)
Cancel recording
Save recording
EDU

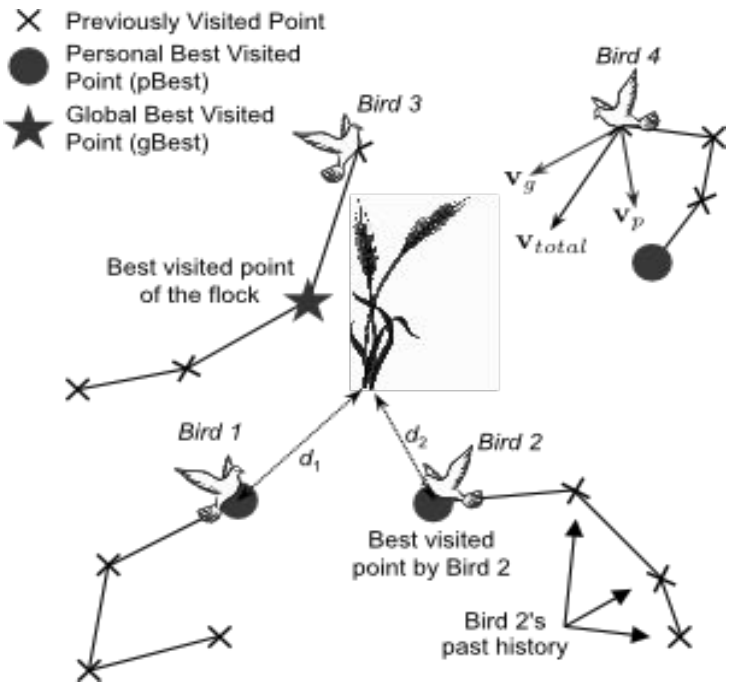
01	When no. for robot increase	• Algorithm become complex
02	What if centralise failed	• Task failure
03	Incrementing/Decrementing	• Have to be monitored

Claim 3. Infact, We need decentralized control!

Algorithm 1.

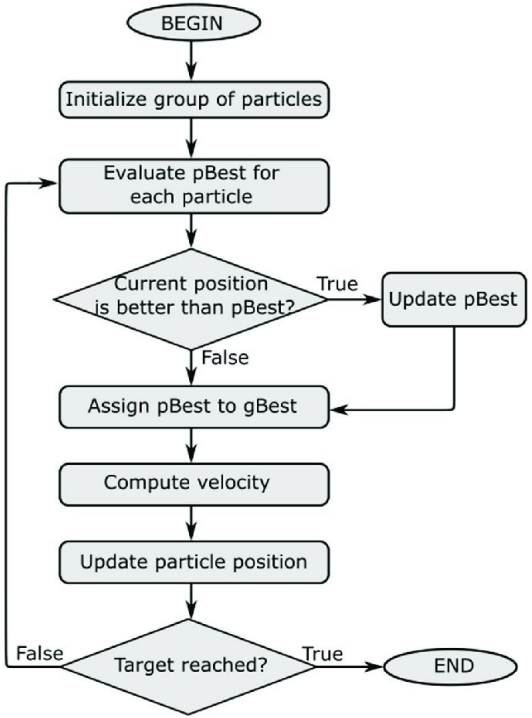
Particle swarm optimization (PSO)





Reference：http://joshkovitz.com/research/projects/optimization-in-electromagnetics/

```
 for each particle i = 1, ..., S do
   Initialize the particle's position with a uniformly distributed random vector: x_i ~ U(b_lo, b_up)
   Initialize the particle's best known position to its initial position: p_i ← x_i
   if f(p_i) < f(g) then
       update the swarm's best known  position: g ← p_i
   Initialize the particle's velocity: v_i ~ U(-|b_up-b_lo|, |b_up-b_lo|)
while a termination criterion is not met do:
   for each particle i = 1, ..., S do
      for each dimension d = 1, ..., n do
         Pick random numbers: r_p, r_g ~ U(0,1)
         Update the particle's velocity: v_i,d ← ω v_i,d + φ_p r_p (p_i,d-x_i,d) + φ_g r_g (g_d-x_i,d)
      Update the particle's position: x_i ← x_i + v_i
      if f(x_i) < f(p_i) then
         Update the particle's best known position: p_i ← x_i
         if f(p_i) < f(g) then
            Update the swarm's best known position: g ← p_i
```

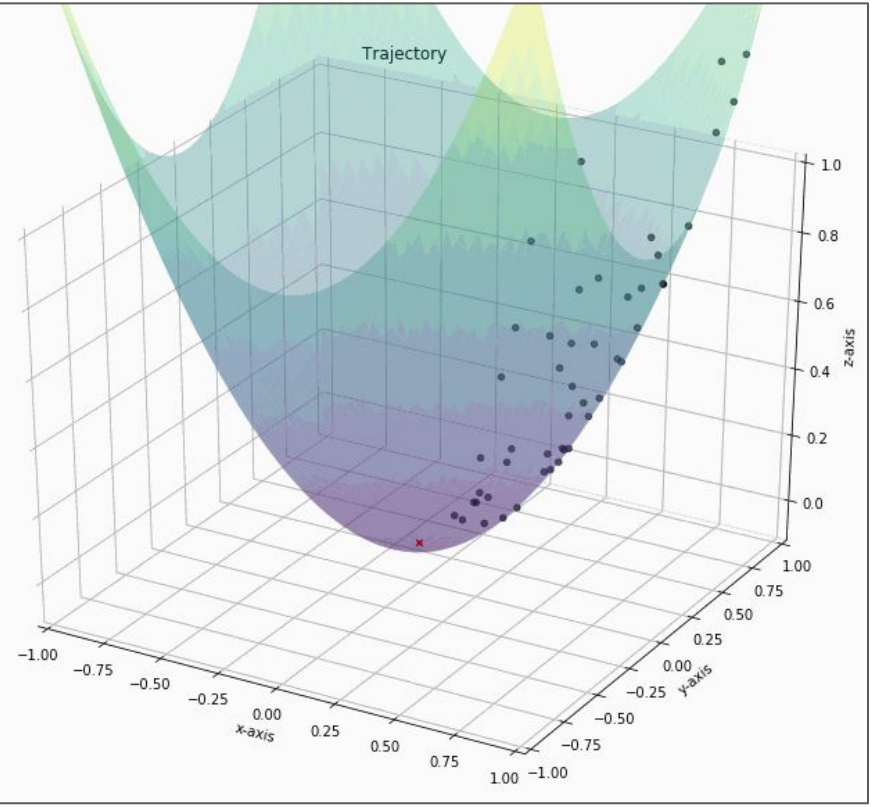


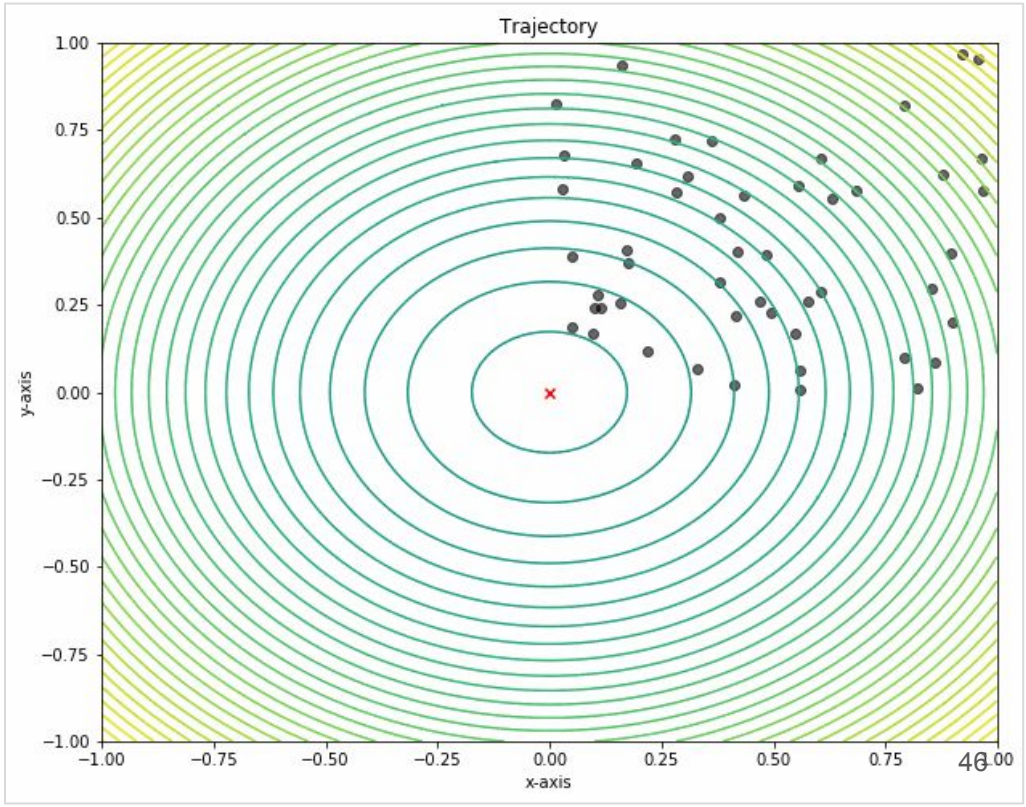
Reference : https://en.wikipedia.org/wiki/Particle_swarm_optimization

Code

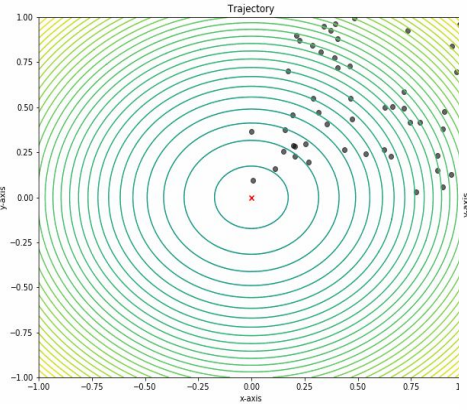
```
while iteration < n_iterations:
    for i in range(n_particles):
        fitness_cadidate = fitness_function(particle_position_vector[i])
        print(fitness_cadidate, ' ', particle_position_vector[i])

        if(pbest_fitness_value[i] > fitness_cadidate):
            pbest_fitness_value[i] = fitness_cadidate
            pbest_position[i] = particle_position_vector[i]

        if(gbest_fitness_value > fitness_cadidate):
            gbest_fitness_value = fitness_cadidate
            gbest_position = particle_position_vector[i]

    if(abs(gbest_fitness_value - target) < target_error):
        break

    for i in range(n_particles):
        new_velocity = (W*velocity_vector[i]) + (c1*random.random()) * (pbest_position[i] - particle_position_vector[i]) + (c2*r
        new_position = new_velocity + particle_position_vector[i]
        particle_position_vector[i] = new_position

    iteration = iteration + 1
```

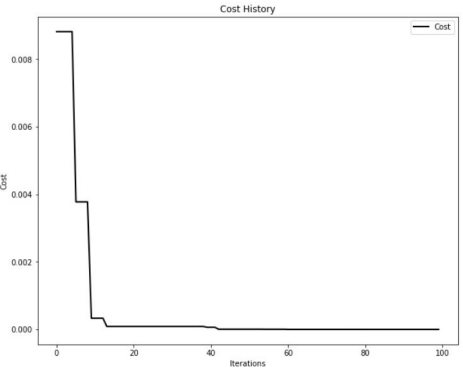
$$cost = x^2 + y^2$$



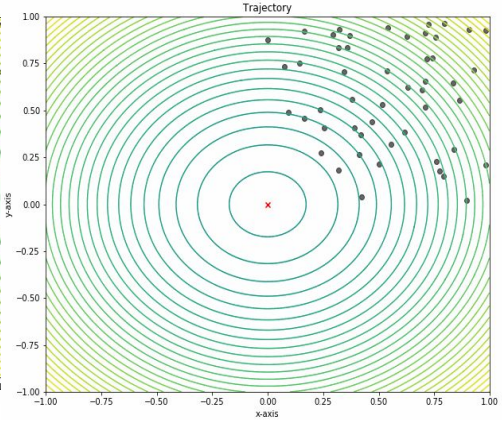


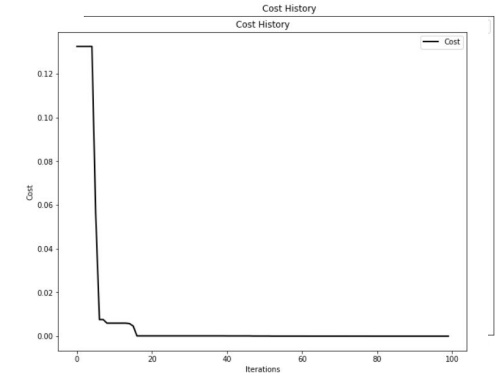
W=0.9, C1=0.5, C2=0.3



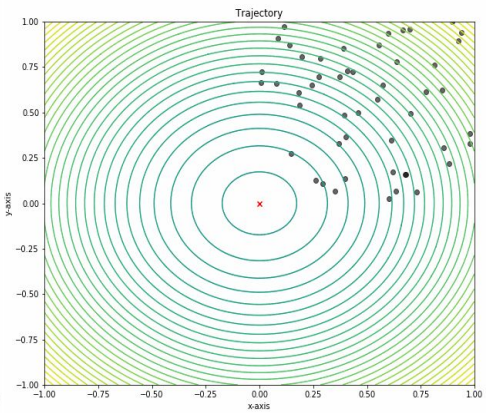


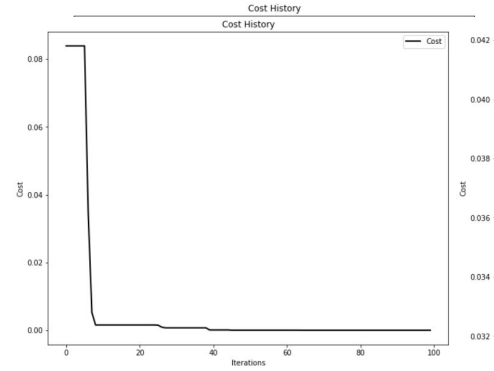
W=0.9, C1=0, C2=0.3



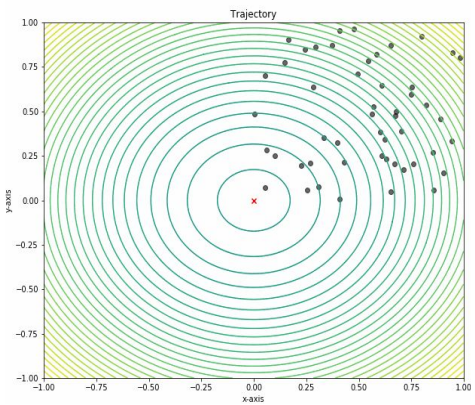


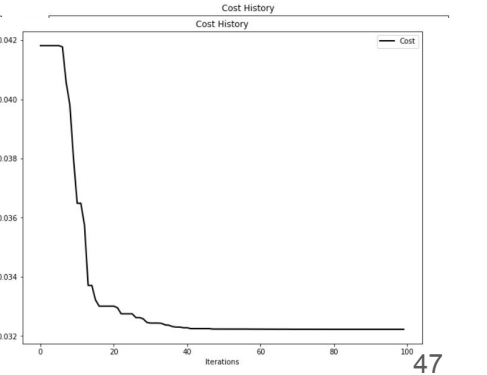
W=0.9, C1=0.5, C2=0



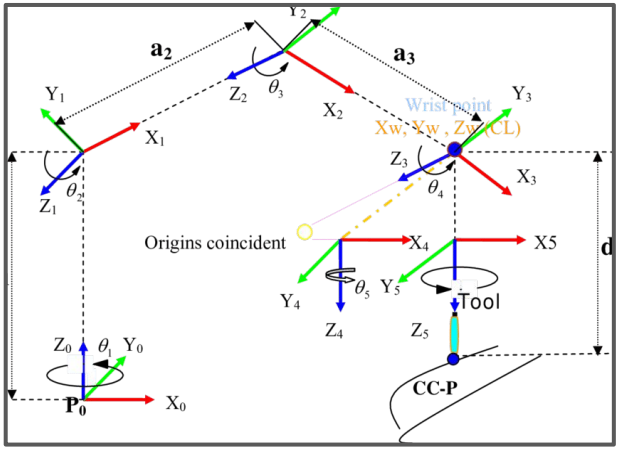


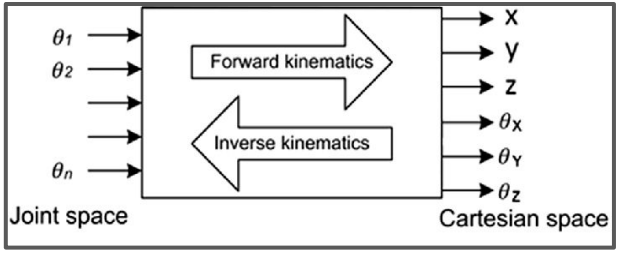
W=0, C1=0.5, C2=0.3





PSO in Inverse kinematics (IK)

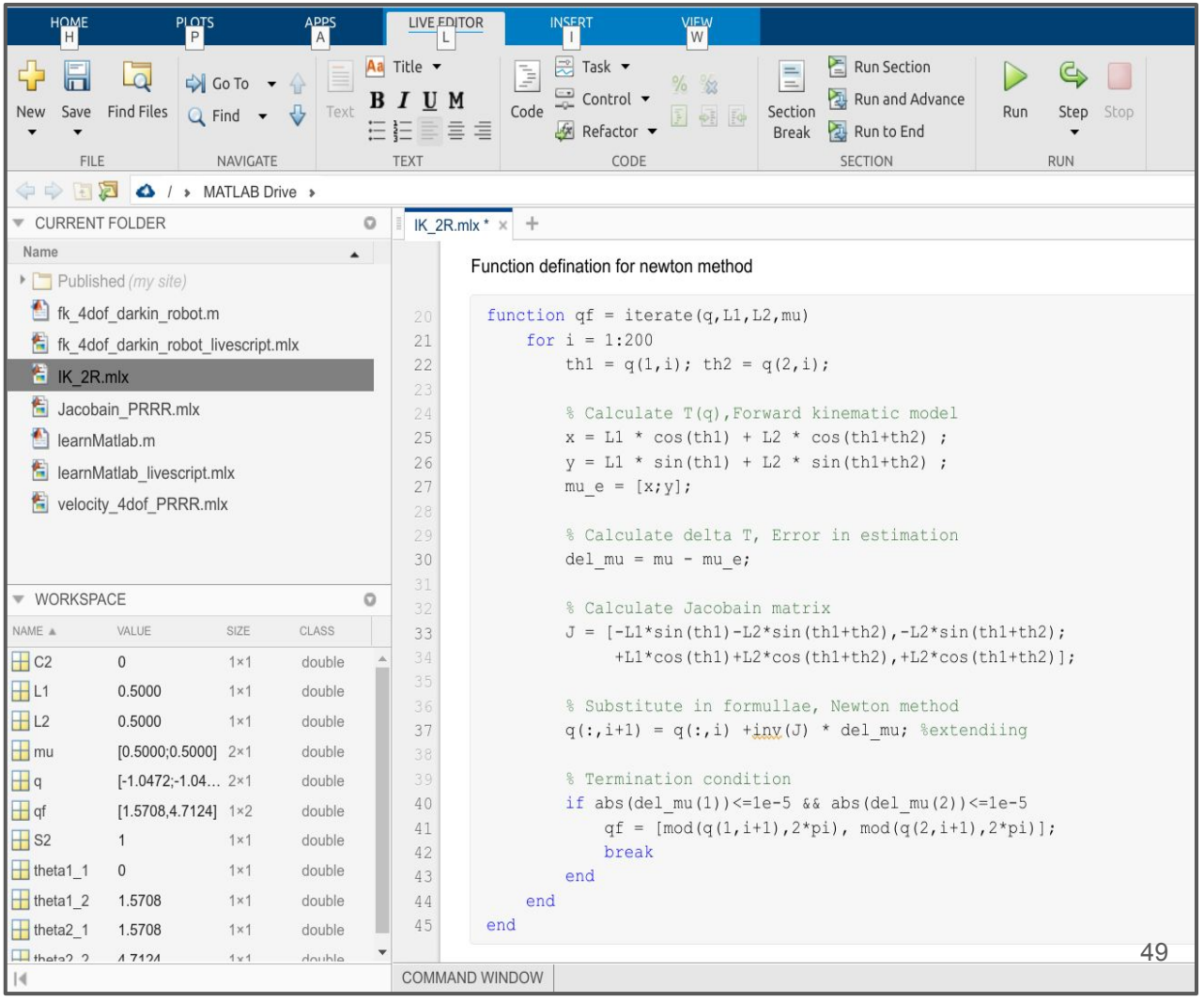




$$\cos q_2 = \frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1 a_2}$$

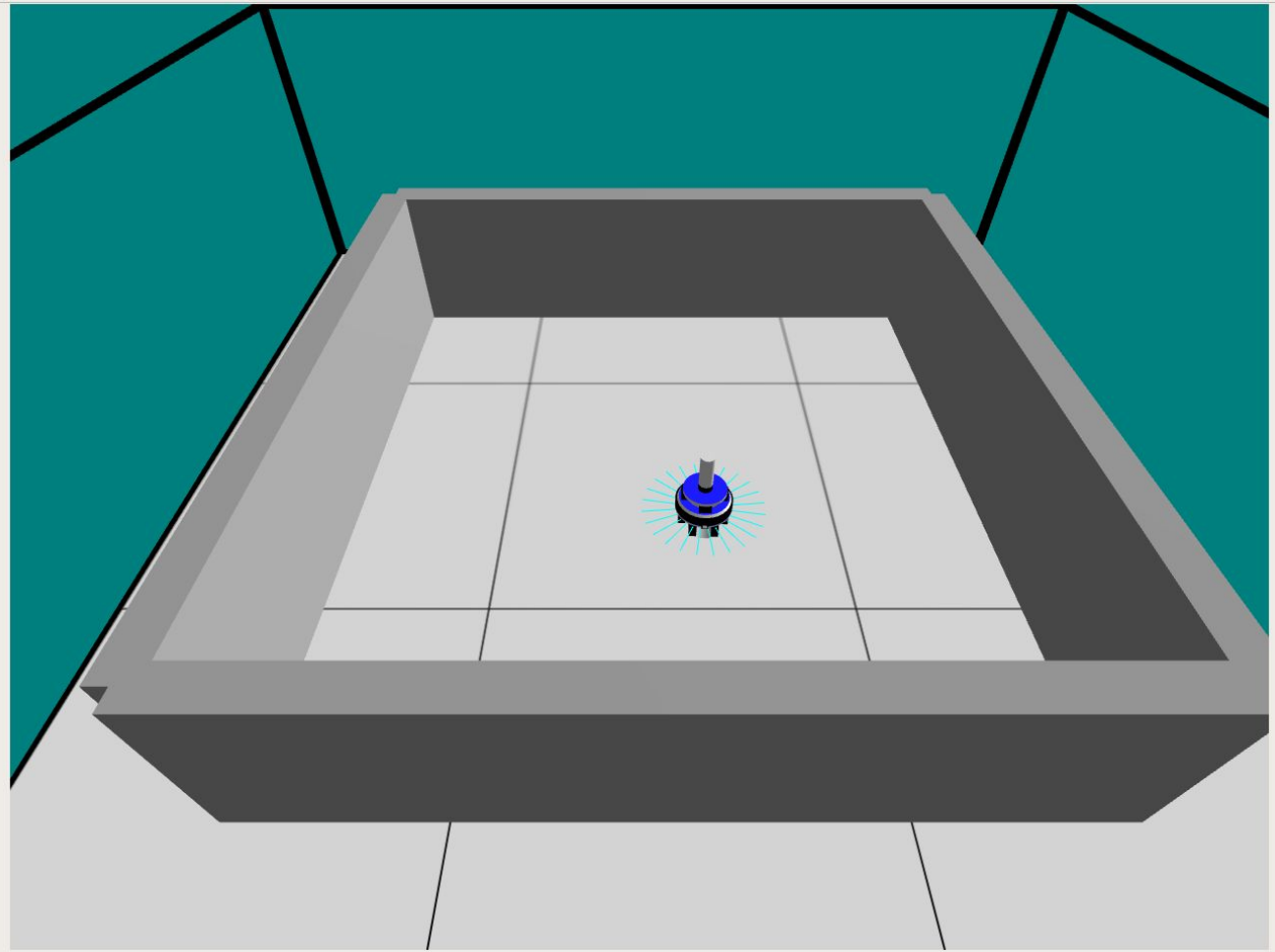
$$q_2 = \cos^{-1} \frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1 a_2}$$

$$q_1 = \tan^{-1} \frac{y}{x} - \tan^{-1} \frac{a_2 \sin q_2}{a_1 + a_2 \cos q_2}$$



Experiment 3. **Foraging**

Experiment Camera ?

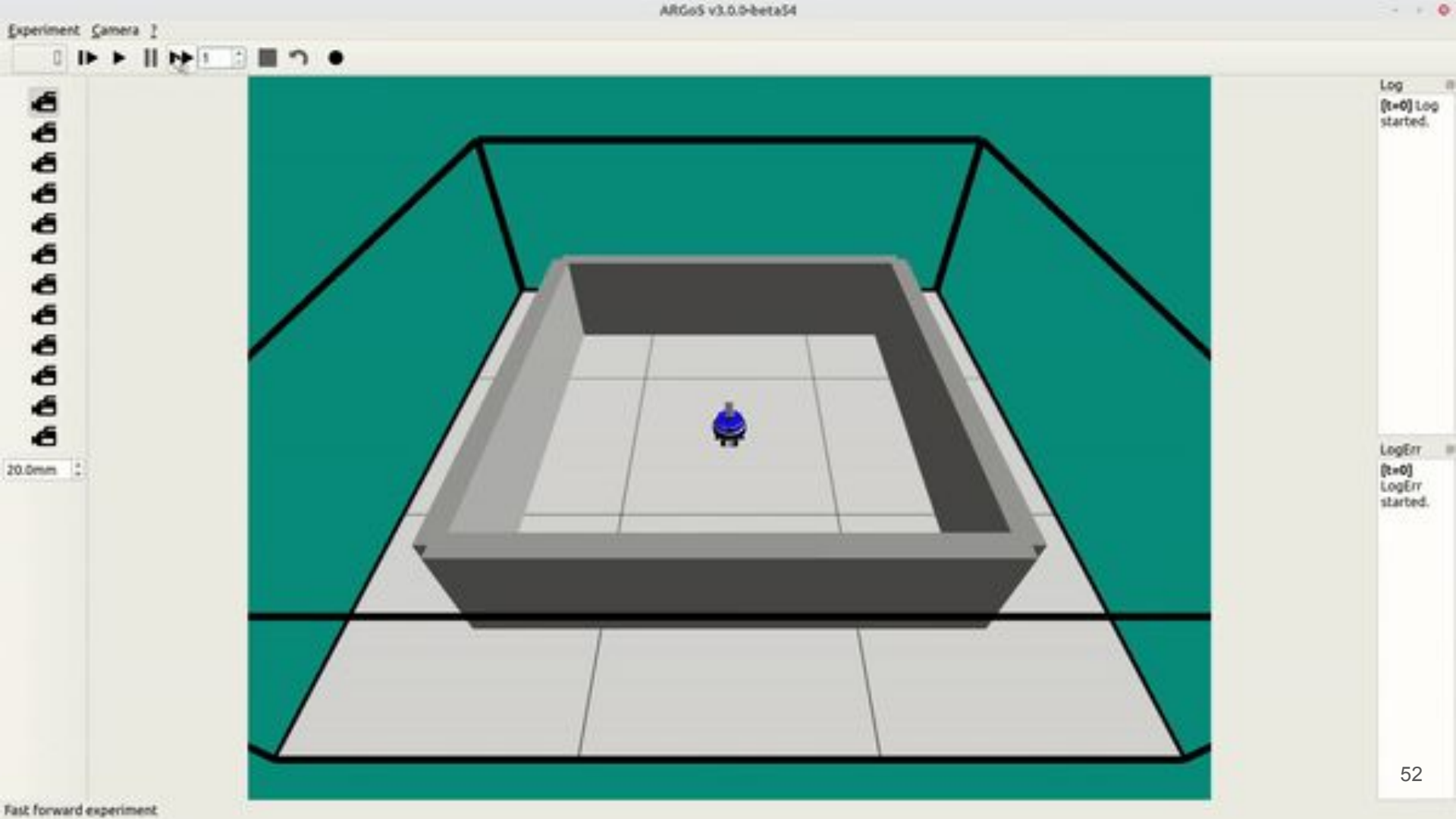


Log

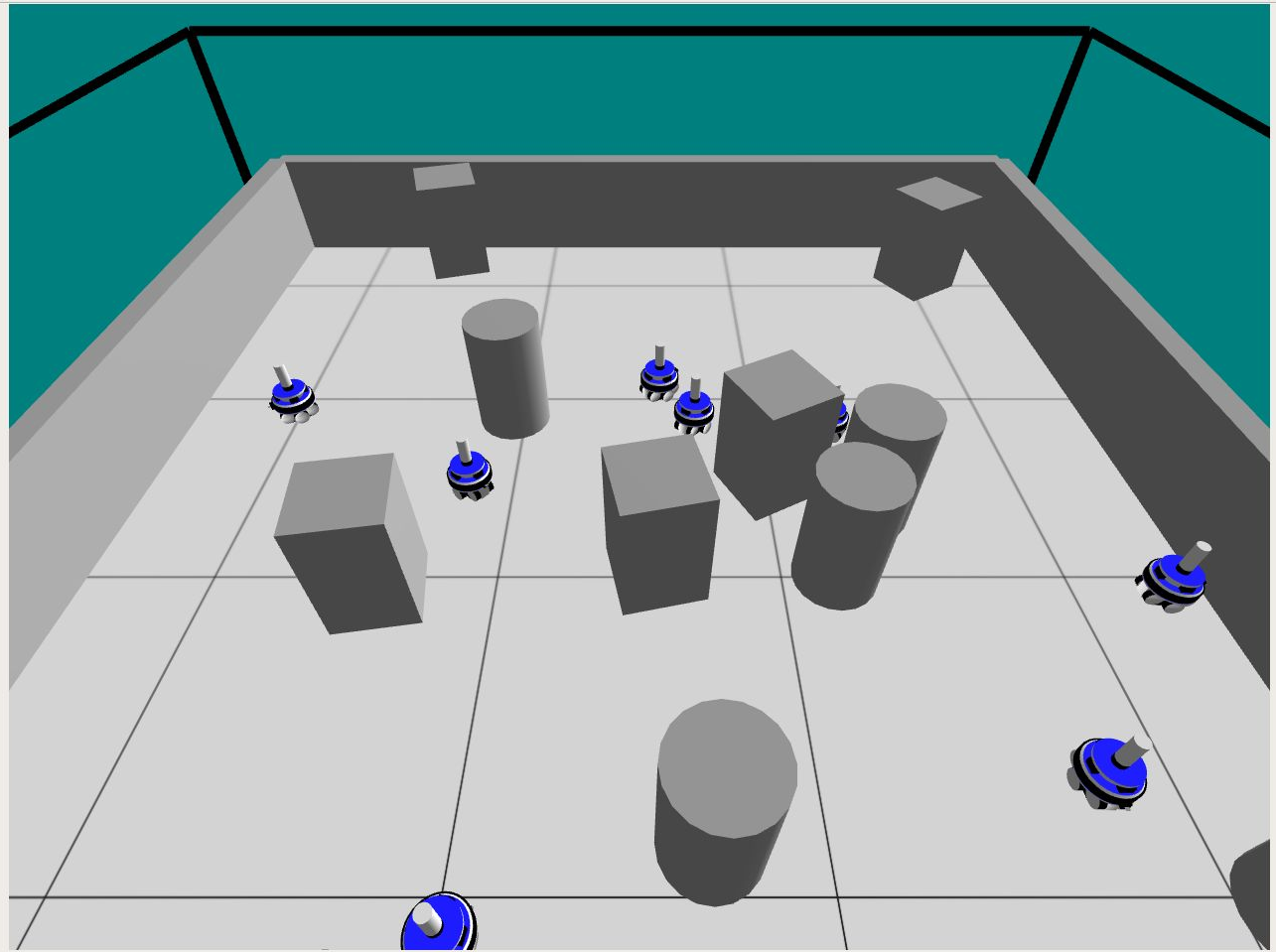
[t=0] Log started.

LogErr

[t=0] LogErr started.


ARGoS v3.0.0-beta54
Experiment Camera ?
20.0mm
Log
[t=0] Log started.
LogErr
[t=0] LogErr started.
Fast forward experiment

Experiment Camera ?



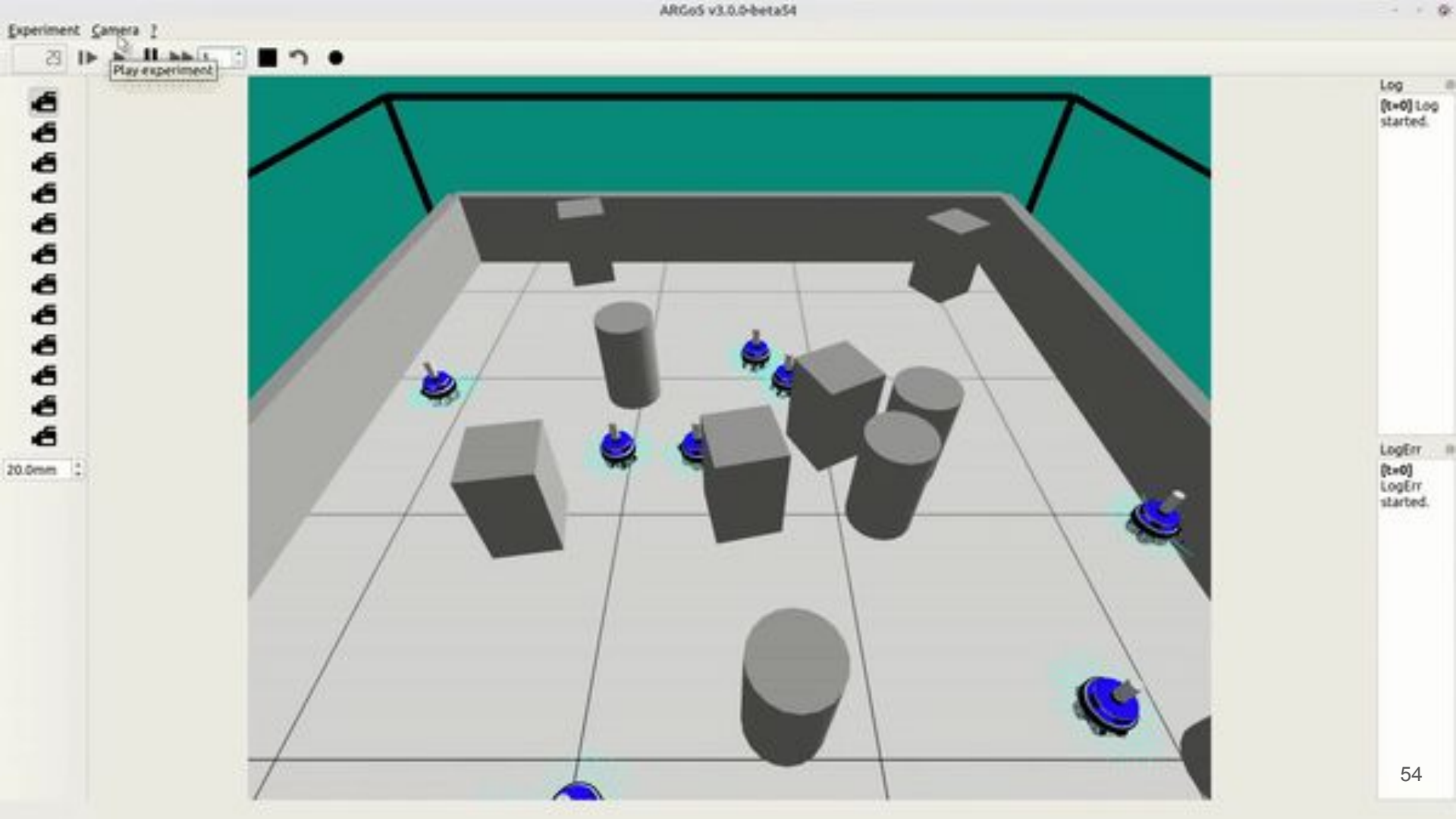
Log

[t=0] Log restarted.

LogErr

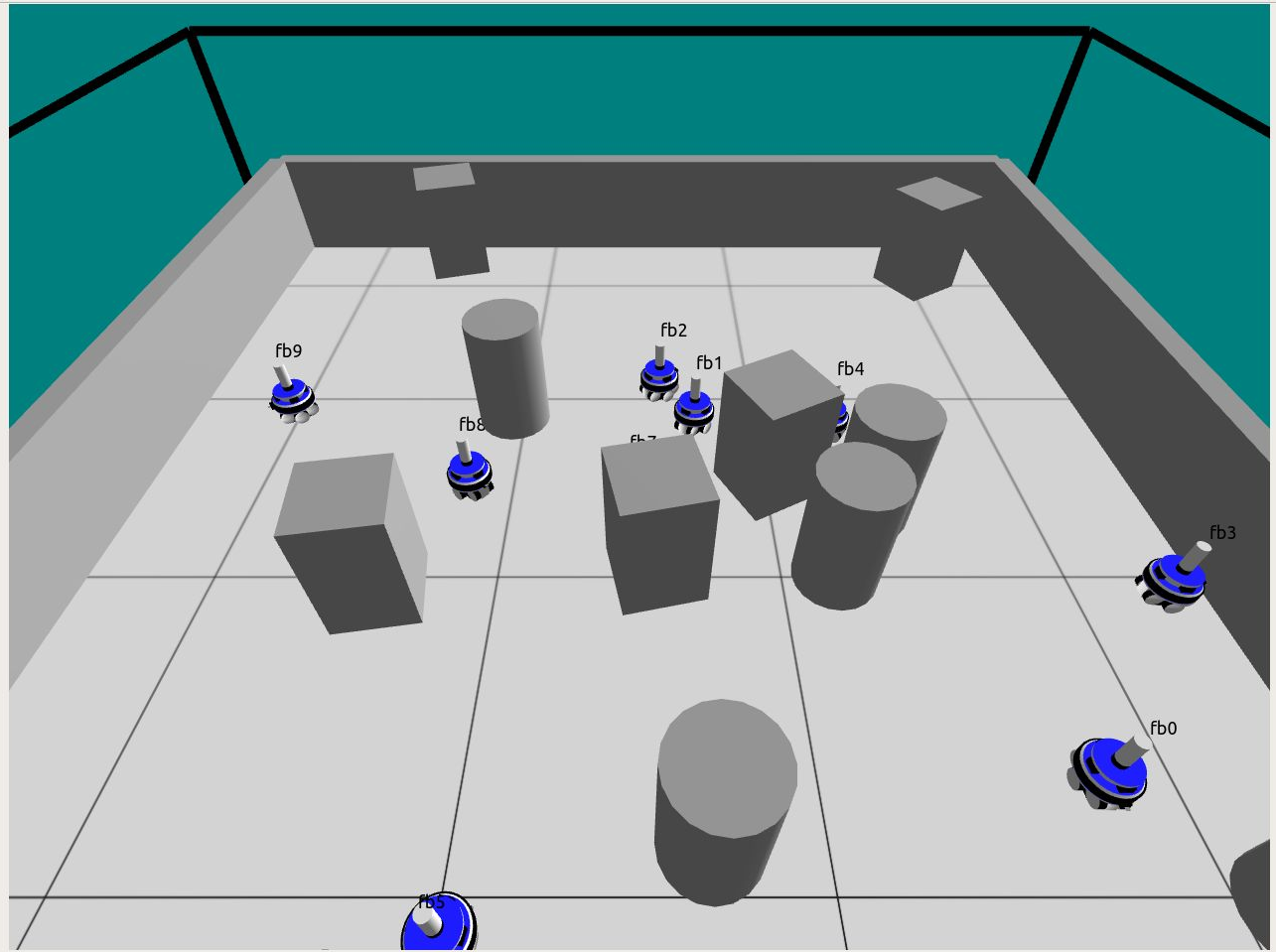
[t=0] LogErr restarted.

20.0mm


ARGoS v3.0.0-beta54
Experiment Camera ?
Play experiment
20.0mm
Log
[t=0] Log started.
LogErr
[t=0] LogErr started.

ARGoS v3.0.0-beta54

Experiment Camera ?



Log

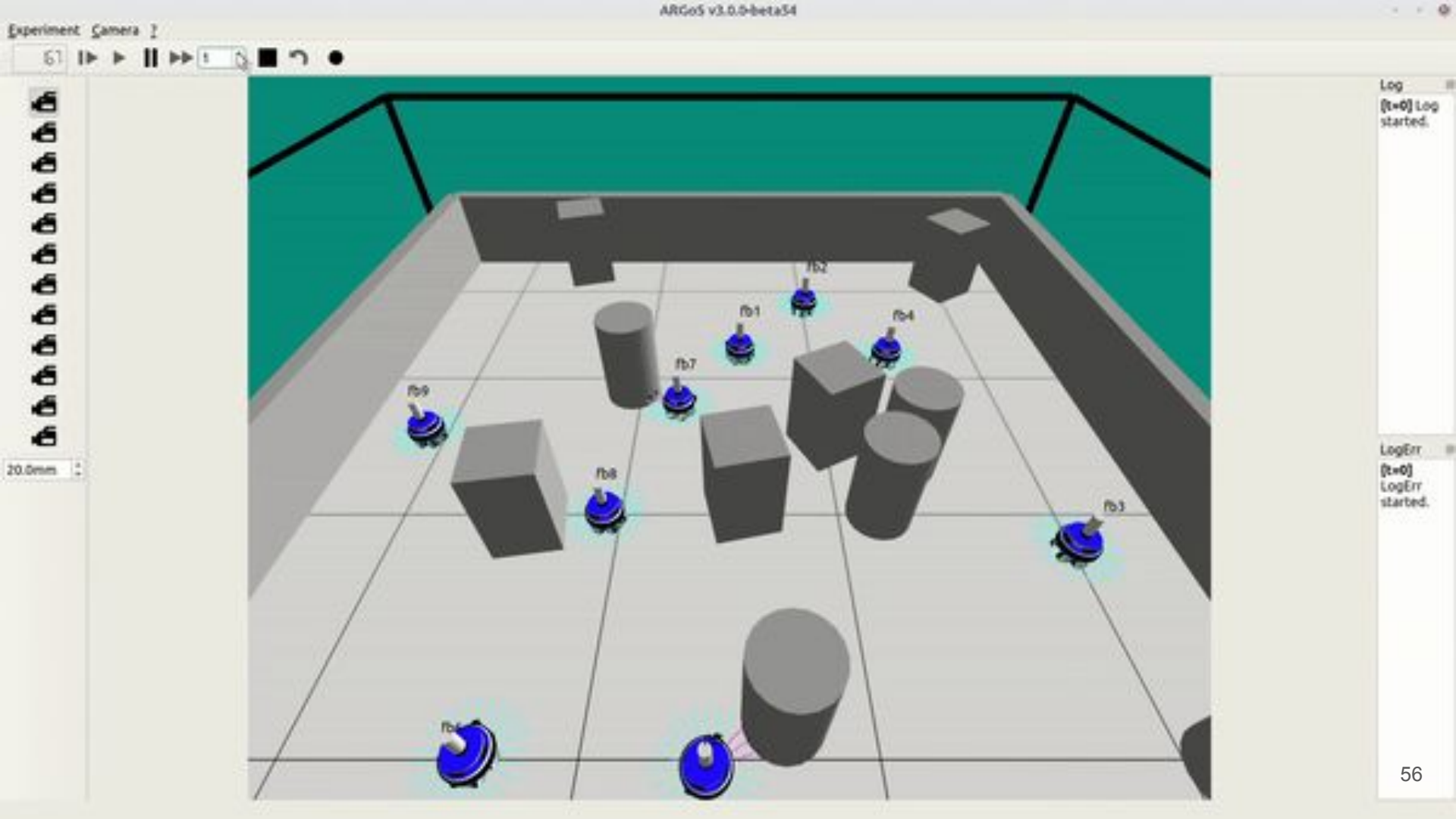
[t=0] Log started.

LogErr

[t=0] LogErr started.

20.0mm

55

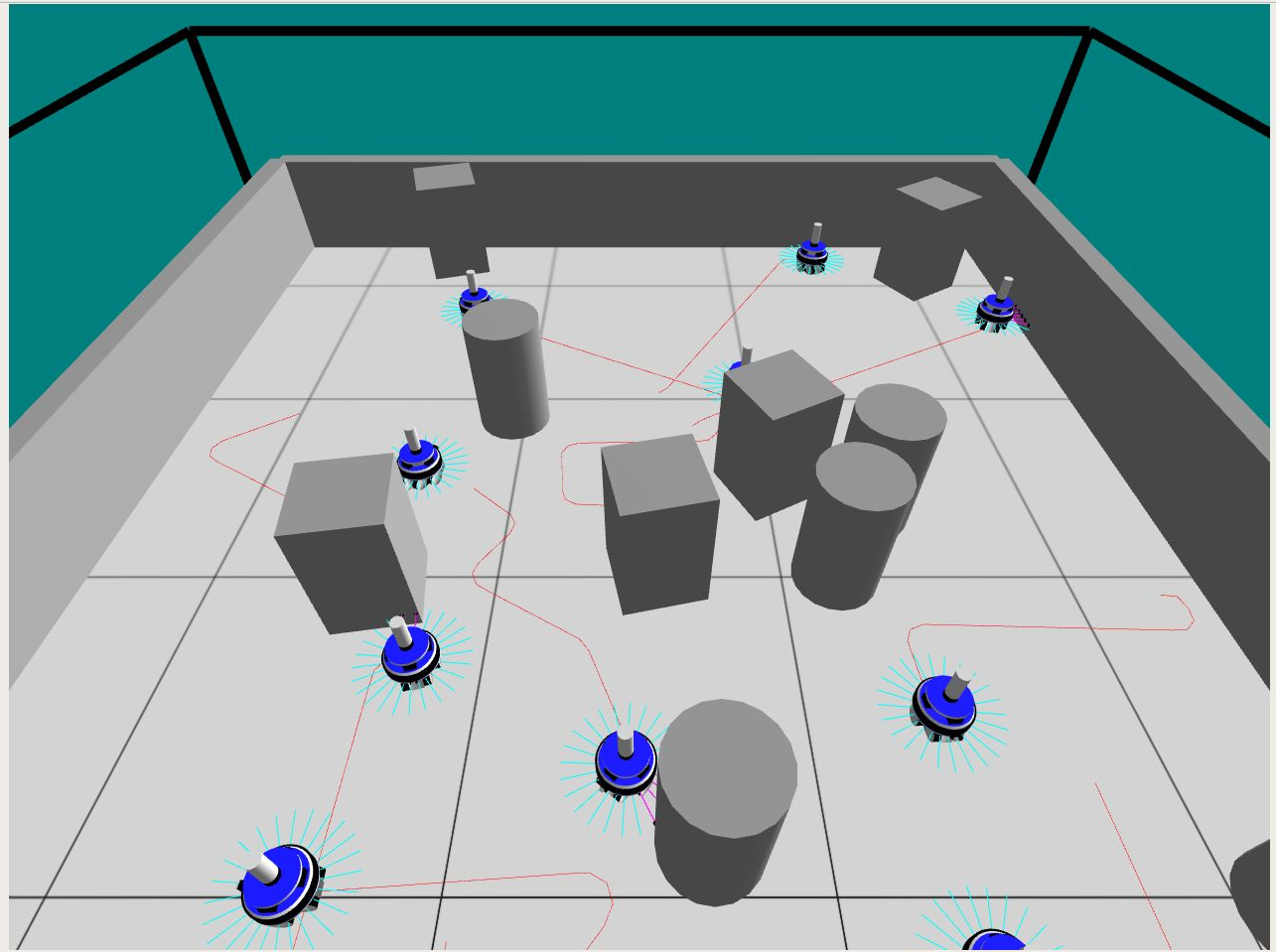

ARGoS v3.0.0-beta54
Experiment Camera ?
fb2
fb1
fb4
fb7
fb9
fb8
fb3
Log
[t=0] Log started.
LogErr
[t=0] LogErr started.

ARGoS v3.0.0-beta54

Experiment Camera ?

371 1

20.0mm



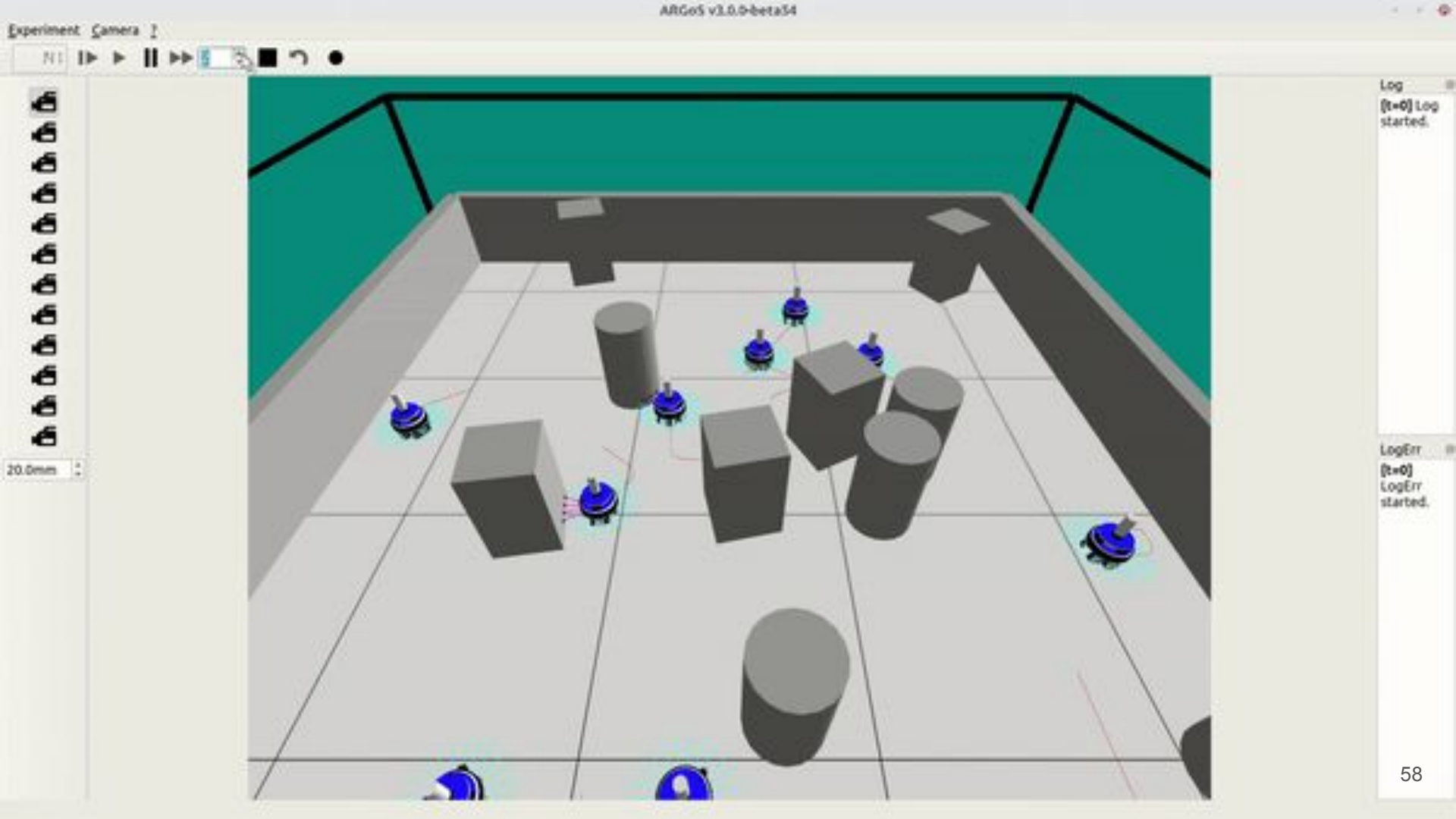
Log

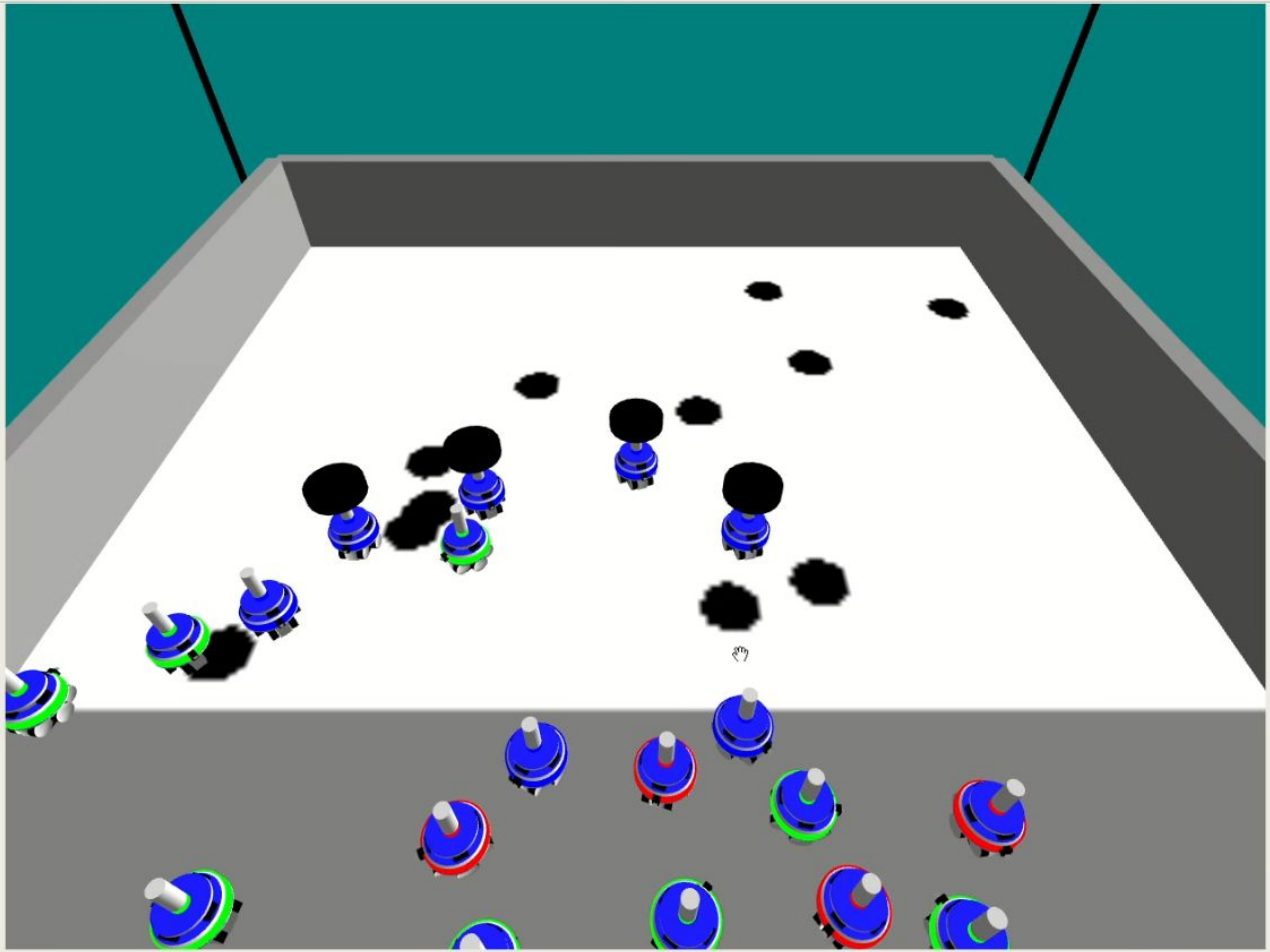
[t=0] Log started.

LogErr

[t=0] LogErr started.

57


ARGoS v3.0.0-beta54
Experiment Camera ?
20.0mm
Log
[t=0] Log started.
LogErr
[t=0] LogErr started.

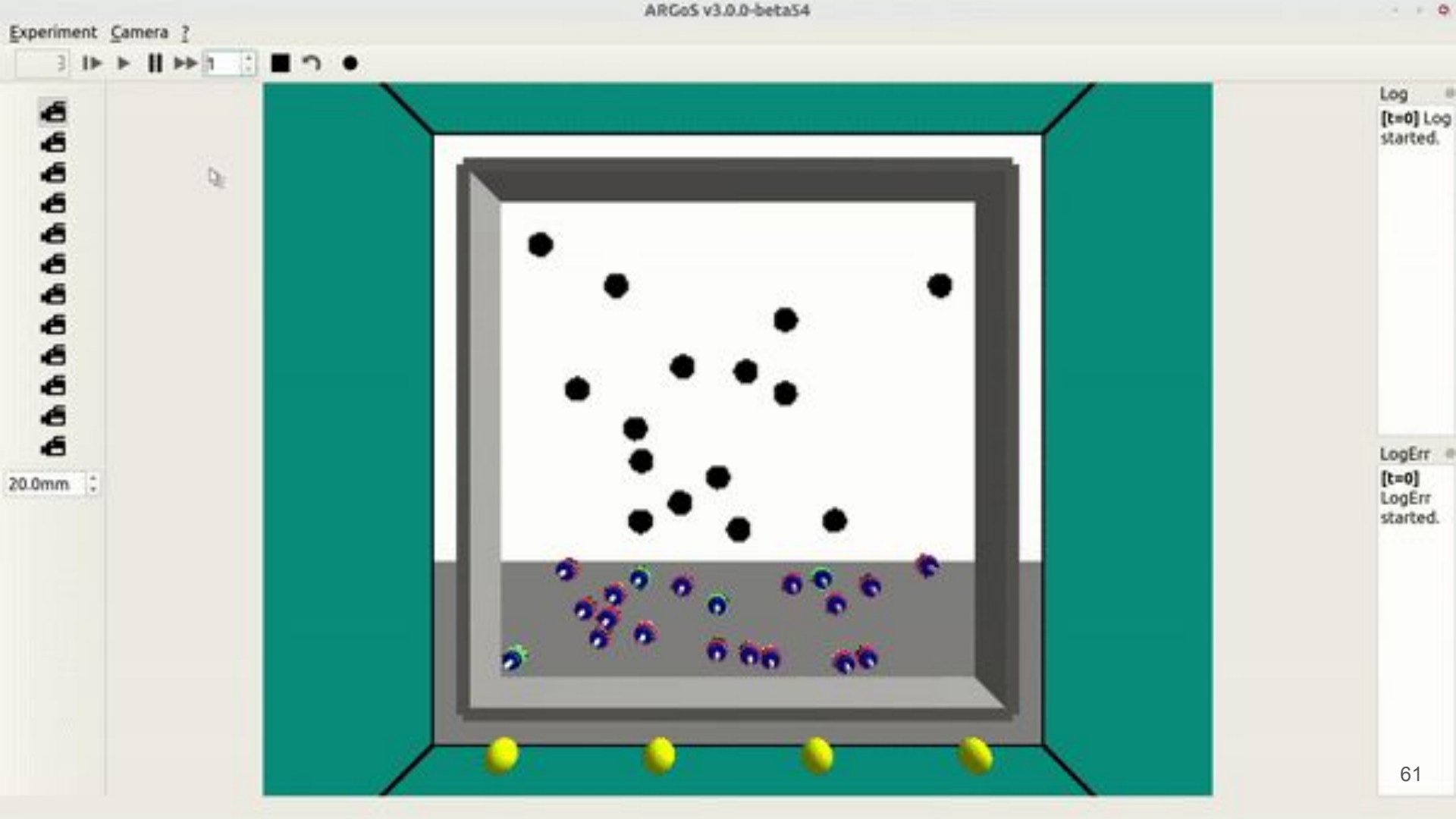


Log
[t=0] Log started.

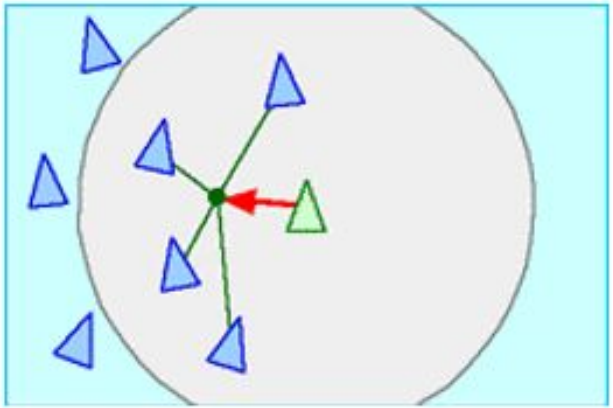
LogErr
[t=0] LogErr started.

Code

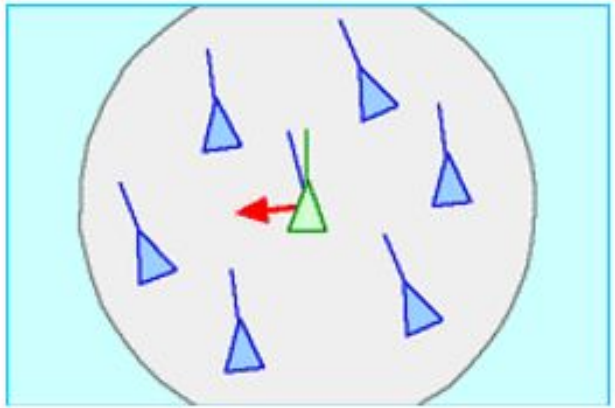
```
// ======================== UpdateState() ========================//
/* Read stuff from the ground sensor */
// ==================== CalculateVectorToLight() ==================//
/* Get readings from light sensor, vector returned points to
* the light thus we know if going toward or away from nest. */
// ==================== DiffusionVector() ==================//
/* Get readings from proximity sensor */
// ==================== SetWheelSpeedsFromVector() ==================//
/* Get the heading angle */
/* Finally, set the wheel speeds */
// ==================== Explore() ==================//
/* We switch to 'return to nest' in two situations:
* 1. if we have a food item
* 2. if we have not found a food item for some time;
/* Switch to 'return to nest' */
// ==================== ReturnToNest() ==================//
/* As soon as you get to the nest, switch to 'resting' */
/* Are we in the nest?*/
/* If then Have we looked for a place long enough? */
/* Yes, stop the wheels... */
/* Tell people about the last exploration attempt and switch to state 'resting' else then, keep looking */
```


ARGoS v3.0.0-beta54
Experiment Camera ?
20.0mm
Log
[t=0] Log started.
LogErr
[t=0] LogErr started.

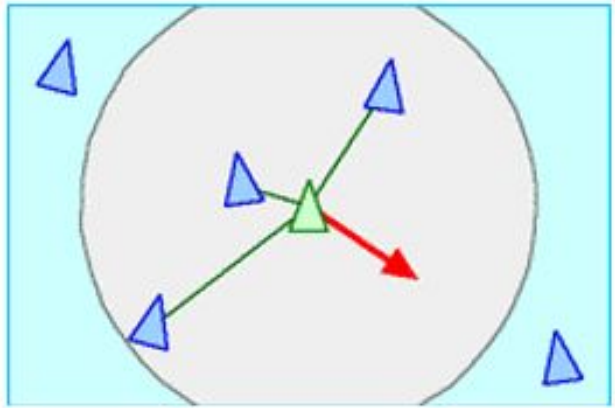
Experiment 4. **Flocking**



Cohesion :
Steer to move towards the average position of the flock's members.



Alignment :
Steer towards the average heading of the flock's members.

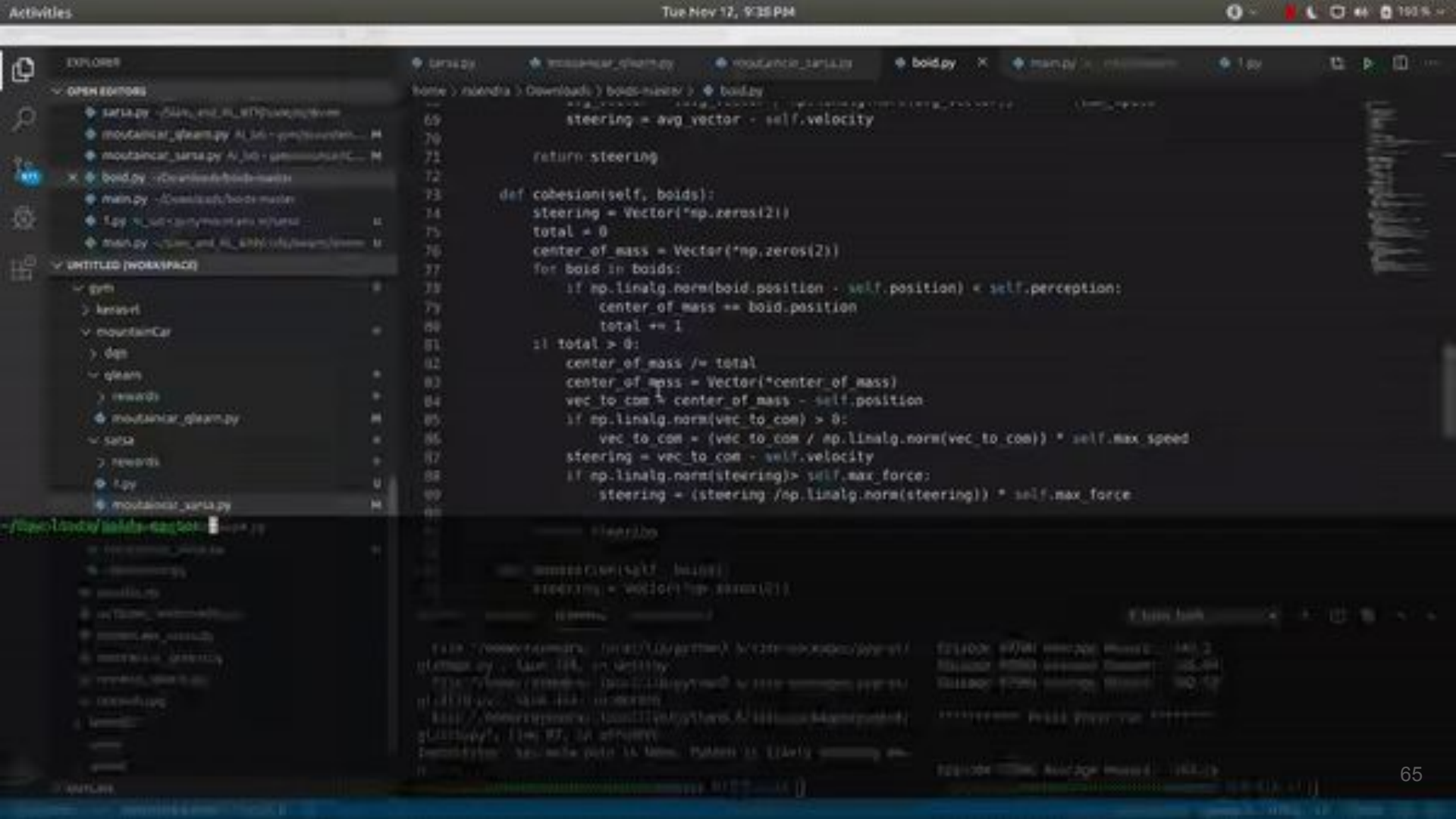


Separation :
Steer to avoid crowding the flock's members.

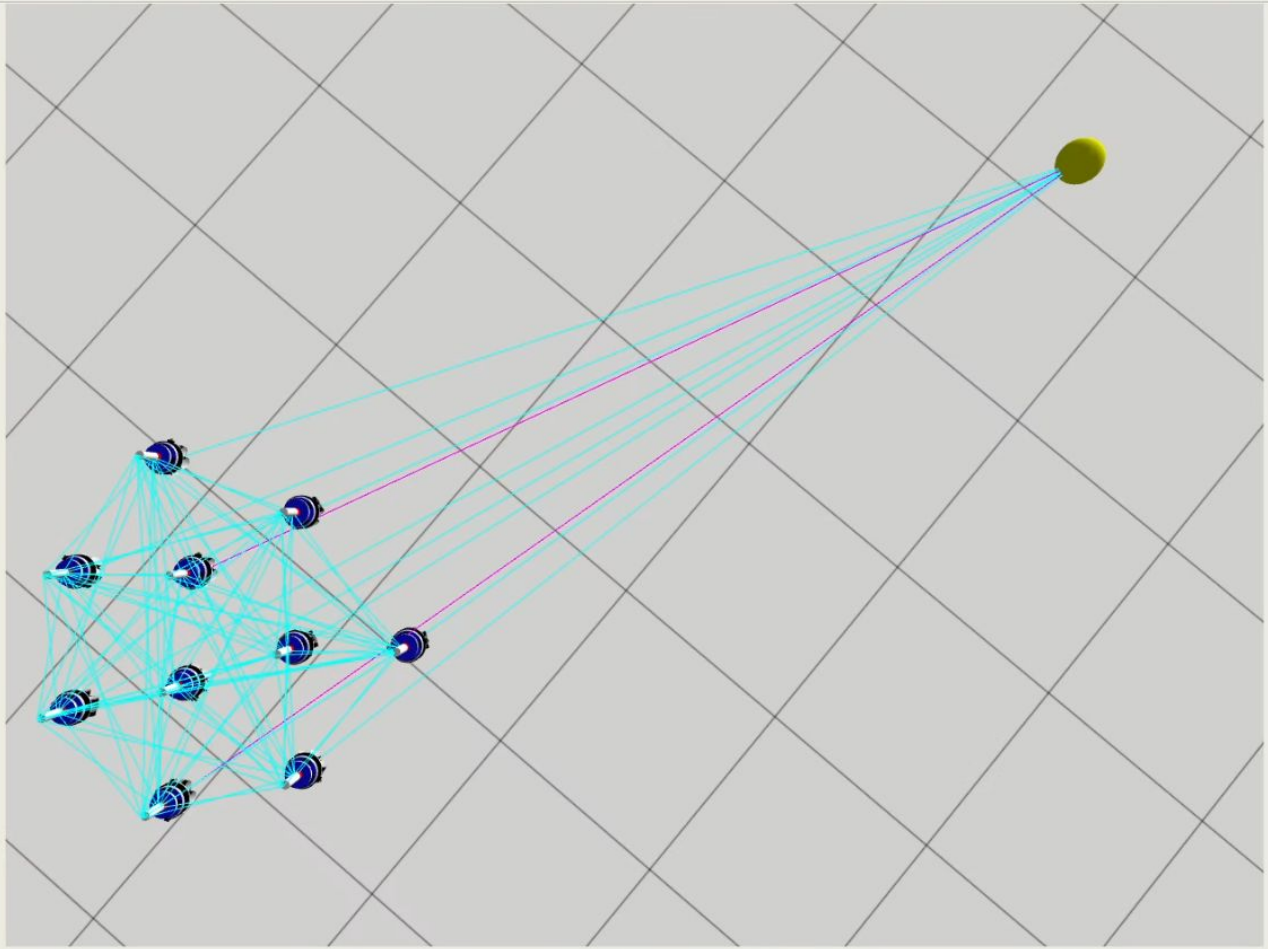
Reference : https://software.intel.com/en-us/articles/an-approach-to-parallel-processing-with-unity

Cohesion code

```
def cohesion(self, boids):
    # initialise
    steering = Vector(*np.zeros(2))
    total = 0
    center_of_mass = Vector(*np.zeros(2))

    # for all particles
    for boid in boids:
        # those who're are within circle
        if np.linalg.norm(boid.position - self.position) < self.perception:
            center_of_mass += boid.position
            total += 1
    if total > 0:
        center_of_mass /= total #com
        center_of_mass = Vector(*center_of_mass)
        vec_to_com = center_of_mass - self.position #vector to com
        # normalise vec_to_com
        if np.linalg.norm(vec_to_com) > 0:
            vec_to_com = (vec_to_com / np.linalg.norm(vec_to_com)) * self.max_speed
        steering = vec_to_com - self.velocity
        # normalise steering
        if np.linalg.norm(steering)> self.max_force:
            steering = (steering /np.linalg.norm(steering)) * self.max_force

    return steering
```


Activities
Tue Nov 12, 9:38 PM
boid.py
steering = avg_vector - self.velocity
return steering
def cohesion(self, boids):
steering = Vector(*np.zeros(2))
total = 0
center_of_mass = Vector(*np.zeros(2))
for boid in boids:
if np.linalg.norm(boid.position - self.position) < self.perception:
center_of_mass += boid.position
total += 1
if total > 0:
center_of_mass /= total
center_of_mass = Vector(*center_of_mass)
vec_to_com = center_of_mass - self.position
if np.linalg.norm(vec_to_com) > 0:
vec_to_com = (vec_to_com / np.linalg.norm(vec_to_com)) * self.max_speed
steering = vec_to_com - self.velocity
if np.linalg.norm(steering)> self.max_force:
steering = (steering /np.linalg.norm(steering)) * self.max_force

Experiment Camera ?



Log

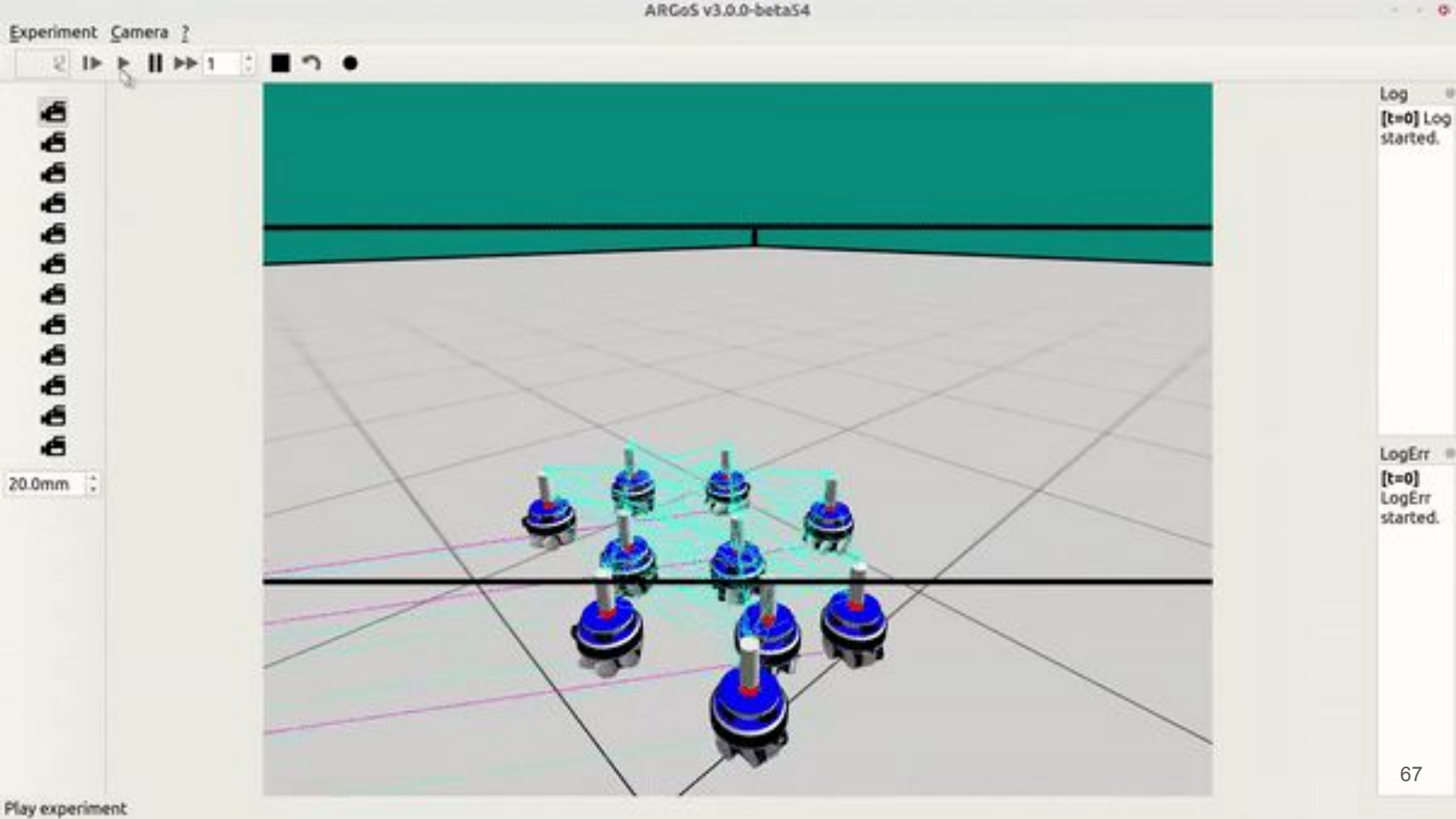
[t=0] Log started.

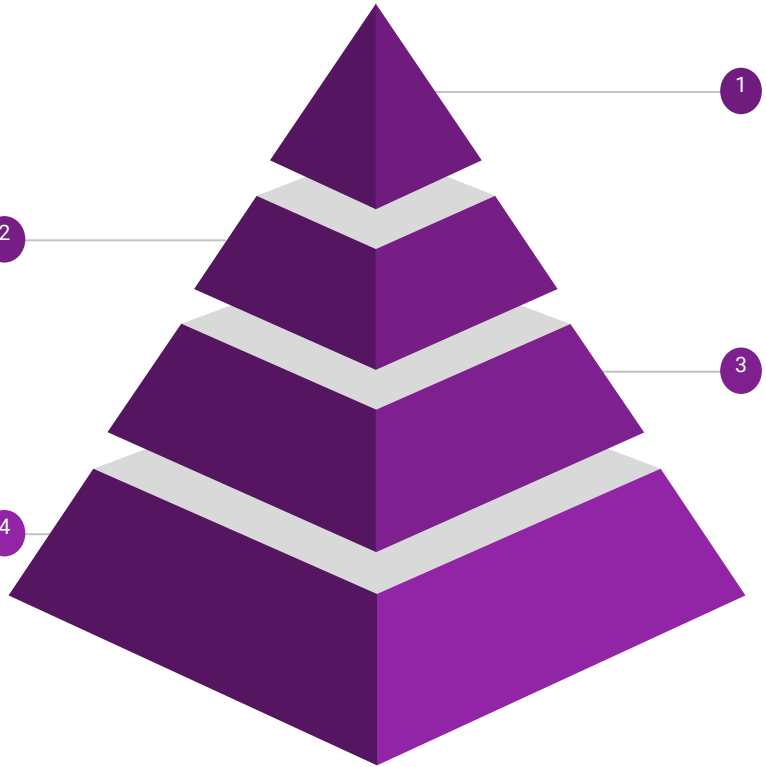
LogErr

[t=0] LogErr started.

20.0mm

Switch to camera 4


ARGoS v3.0.0-beta54
Experiment Camera ?
1
20.0mm
Log
[t=0] Log started.
LogErr
[t=0] LogErr started.
Play experiment



Hardware Implementation
✘ ✘

Coordination ✓ ✓

Swarm Intelligence, Flocking, Foraging

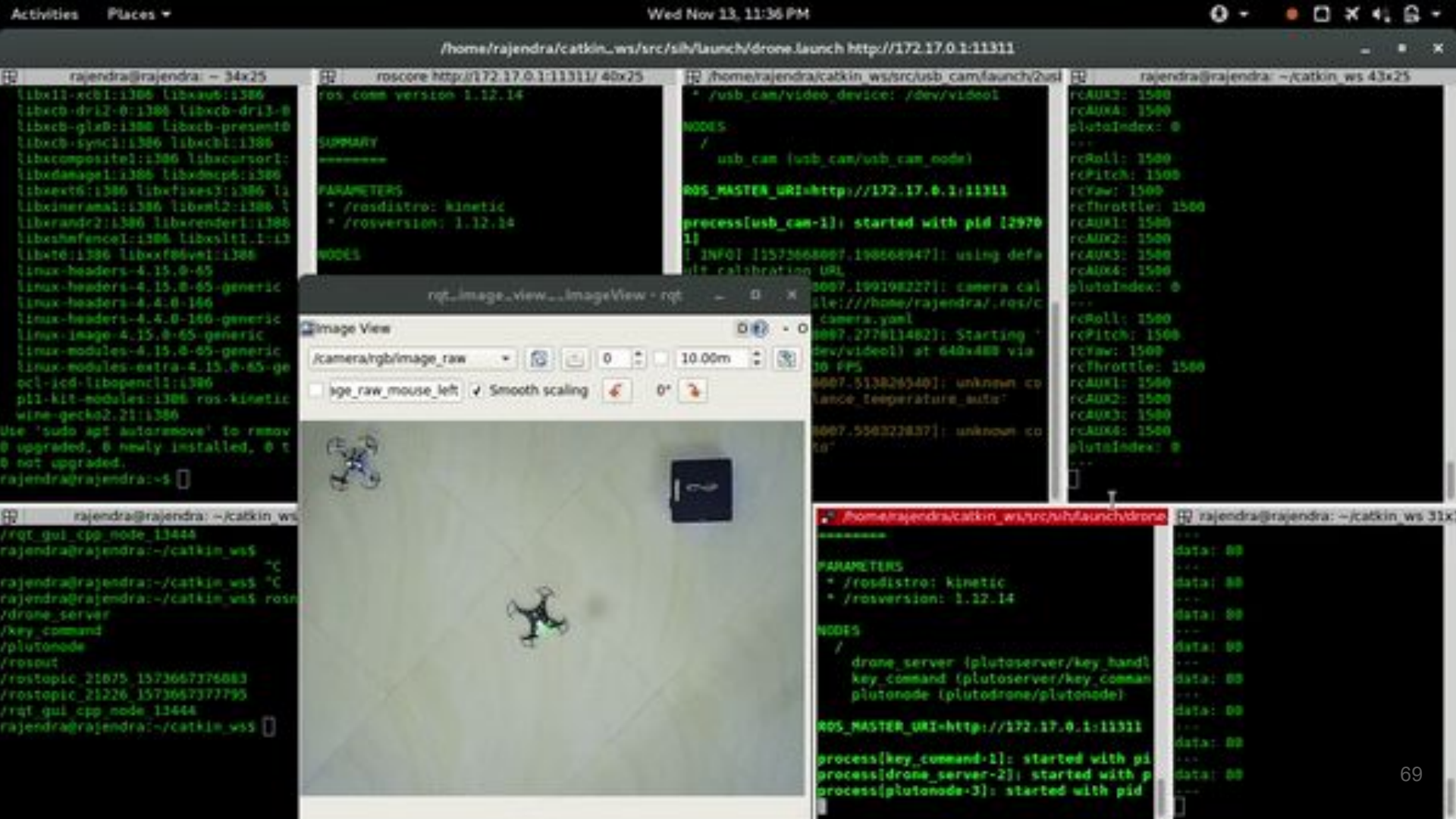
Control ✓ ✓

Path planning, PID, RL, A*.

Perception ✓ ✓

Slam algorithms - Gmapping, RTABMAP, Hector, Karto. (3D cameras, Lidars).

SLAM


Activities
Places
Wed Nov 13, 11:36 PM
/home/rajendra/catkin_ws/src/sih/launch/drone.launch http://172.17.0.1:11311
rajendra@rajendra: ~ 34x25
roscore http://172.17.0.1:11311/ 40x25
/home/rajendra/catkin_ws/src/usb_cam/launch/usb
rajendra@rajendra: ~/catkin_ws 43x25
rqt_image_view__ImageView - rqt
Image View
/camera/rgb/image_raw
Smooth scaling
rajendra@rajendra: ~/catkin_ws
/home/rajendra/catkin_ws/src/sih/launch/drone
rajendra@rajendra: ~/catkin_ws 31x

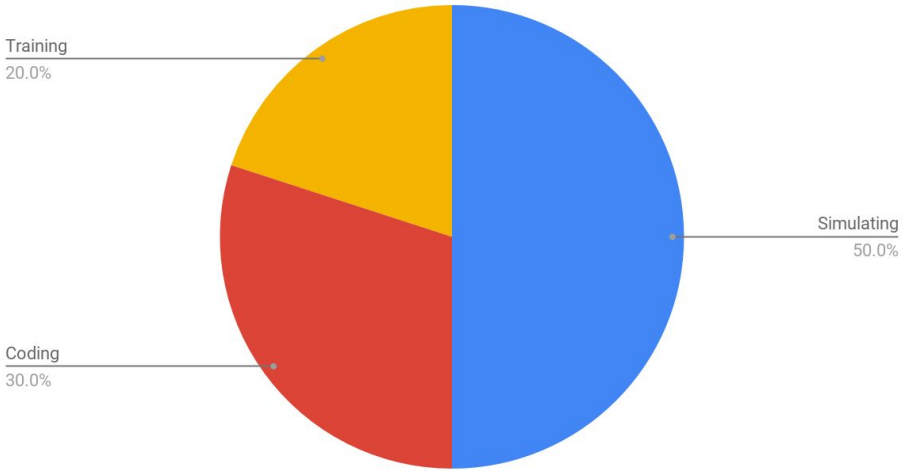
3. Conclusion

Progress

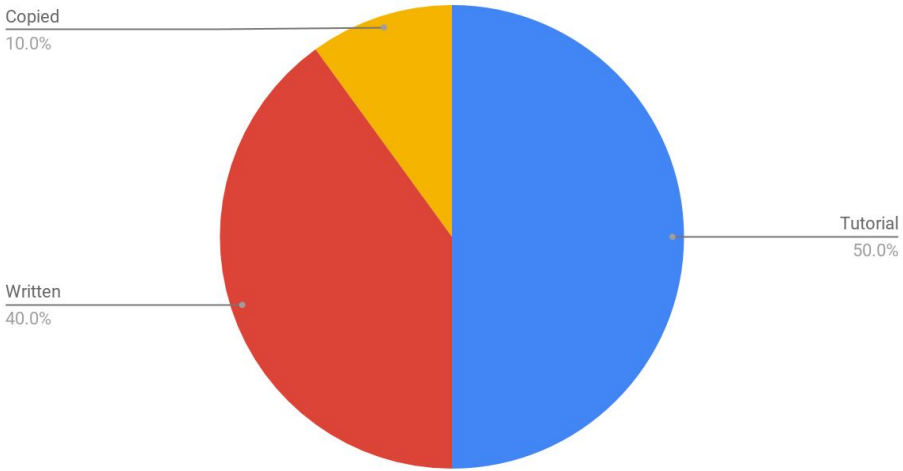
Perception ✓✓	Control ✓✓	Coordination ✓✓	Hardware Implementation ✘✘
Done - Feature Detector(ICP, SIFT, SURF),Gmapping, RTABMAP, 3D reconstruction and processing, Feature matching	**Done** - PID, Q-learning, Sarsa,DQN	**Done** - PSO, flocking, foraging	**Done** - Perception, hovering
Not done - Detection, Recognition	**Not done** - DQN variants (DDQN,Priortised DQN, Dueling DQN, A3C, Distributionnal DQN, Noisy DQN, Rainbow)	**Not done** - formation, ACO, MSO	**Not done** - Control, coordination

Effort / Time

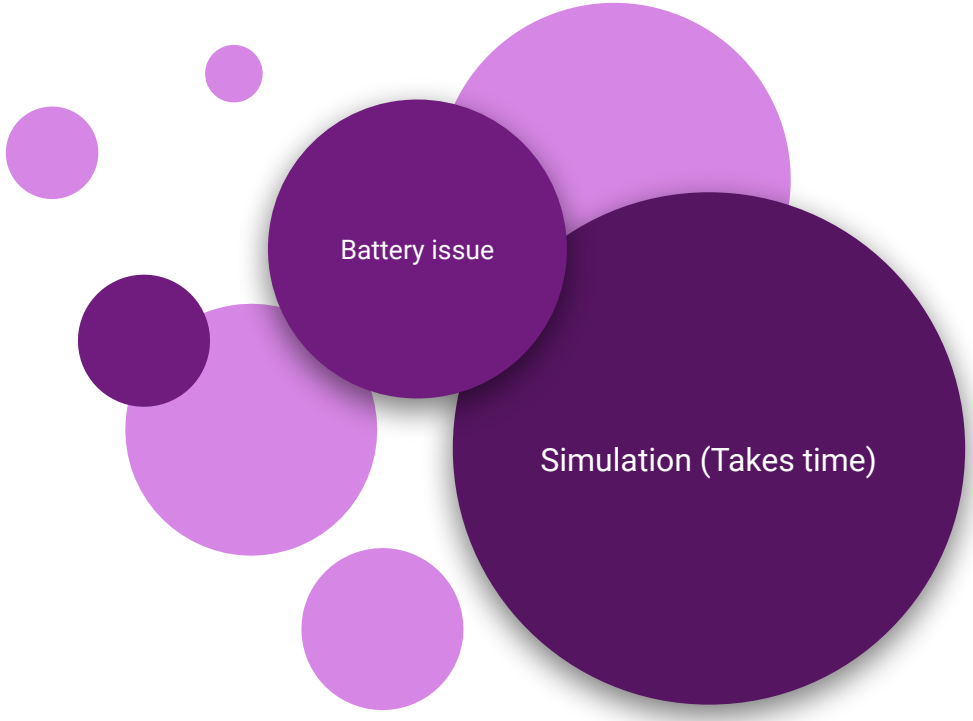
Time



Code



Challenges



THANK YOU