Simple Foraging and Random Aggregation Strategy for swarm robotics without communication

Paulo Roberto Loma Marconi[1]

***Abstract*— In swarm robotics Foraging and Aggregation are basic tasks yet that can be challenging when there is no communication between the robots. This paper proposes a strategy using a Mealy Deterministic Finite State Machine (DFSM) that switches between five states with two different algorithms, the Rebound avoider/follower through proximity sensors, and the Search blob/ePuck using the 2D image processing of the ePuck embedded camera. Ten trials for each scenario are simulated on V-rep in order to analyse the performance of the strategy in terms of the mean and standard deviation.**

I. FORAGING AND RANDOM AGGREGATION

The DFSM diagram in Fig. 1, which is defined by (1), starts in the Behaviour state where the robot is set as *avoider* while the time simulation is $t \leq 60[s]$. During that time, the Foraging state looks for the green blobs with the Search blob/ePuck algorithm while avoiding obstacles using the Rebound algorithm. Moreover, a Random Movement state is used to introduce randomness to the system so the agent can take different paths if there is no blob or obstacle detection. For $60 < t \leq 120$, the Behaviour of the robot is set to *follower* and switches to Random Aggregation state where it uses both algorithms, the Rebound to follow ePucks with the proximity sensors and the Search to look for the closest ePuck wheels. For both algorithms, the output is the angle of attack α_n, where n depends on the current state.

$$\begin{aligned} S &= \{B, F, R, A, Ra\} \\ \Sigma &= \{t \leq 60, 60 < t \leq 120, bl\ \exists, bl\ \nexists, ob\ \exists, ob\ \nexists, \\ &\qquad eP\ \exists, eP\ \nexists\} \\ s_0 &= \{B\} \end{aligned} \tag{1}$$

where, S is the finite set of states, Σ is the input alphabet, $\delta : S \times \Sigma$ is the state transition function, s_0 is the initial state, $\exists$ and $\nexists$ mean detection and no detection respectively.

TABLE I: State transition function δ

Input	Current State	Next State	Output
$t \leq 60$	**B**ehaviour	**F**oraging	avoider
$60 < t \leq 120$	**B**ehaviour	**A**ggregation	follower
blob $\exists$	**F**oraging	**F**oraging	α_C
blob $\nexists$	**F**oraging	**Ra**ndom Mov.	α_{C_r}
obstacle $\exists$	**F**oraging	**R**ebound	α_R
obstacle $\nexists$	**R**ebound	**F**oraging	-
obstacle $\exists$	**A**ggregation	**R**ebound	α_R
obstacle $\nexists$	**R**ebound	**A**ggregation	-
ePuck $\exists$	**A**ggregation	**A**ggregation	α_e
ePuck $\nexists$	**A**ggregation	**Ra**ndom Mov.	α_{e_r}

[1]The author is with the Department of Automatic Control Systems Engineering, The University of Sheffield, UK prlomamarconi1@sheffiel.ac.uk

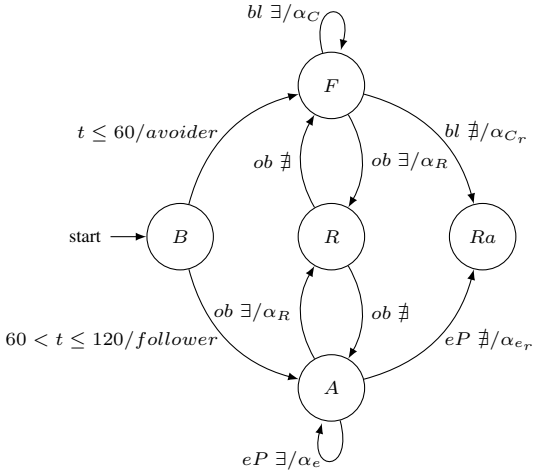


Fig. 1: Mealy DFSM of the controller

II. IMPLEMENTATION

A. Unicycle model

The Unicycle model in Fig. 2a [1] controls the angular velocities of the right and left wheels, v_r and v_l as follows,

$$\begin{aligned} v_r &= (2\ V_x + \omega\ L)/(2\ R) \\ v_l &= (2\ V_x - \omega\ L)/(2\ R) \end{aligned} \tag{2}$$

where, V_x is the linear velocity of the robot, L is the distance between the wheels, R is the radius of each wheel, and ω is the angular velocity of the robot. Using α_n and the simulation sampling period T, the control variable for the simulation is $\omega = \alpha_n/T$, refer to code line 23, 196 and 214.

B. Rebound avoider/follower algorithm

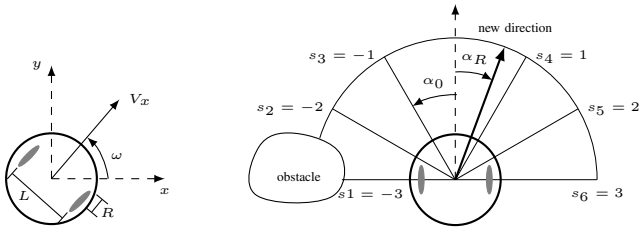
The Rebound algorithm [2] calculates the Rebound angle α_R to avoid/follow an obstacle/objective given $\alpha_0 = \pi/N$ and $\alpha_i = i\ \alpha_0$,

$$\alpha_R = \frac{\sum_{i=-N/2}^{N/2}\ \alpha_i\ D_i}{\sum_{i=-N/2}^{N/2}\ D_i} \tag{3}$$

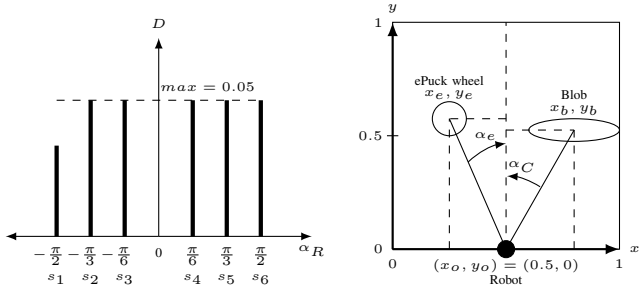
where, α_0 is the uniformly distributed angular pace, N is the number of sensors, α_i is the angular information per pace $\alpha_i\ \epsilon\ \left[-\frac{N}{2}, \frac{N}{2}\right]$, and D_i is the distance value obtained by the proximity sensors, refer to code line 17 and 138.

The weight vector given by α_i sets the robot behaviour for each corresponding mapped sensor $\{s_1, s_2, s_3, s_4, s_5, s_6\}$. For the *avoider* is $\{-3, -2, -1, 1, 2, 3\}$, and for the *follower* is $\{3, 2, 1, -1, -2, -3\}$. Fig. 2b and Fig. 2c show an example

of α_R with the Vector Field Histogram (VFH) for the *avoider* case. Refer to code line 127 and 131.



(a) Unicycle (b) Rebound angle (avoider)



(c) Avoider VFH (d) Blob and ePuck angle

Fig. 2: Unicycle model, Rebound and Search angle

C. Search blob/ePuck algorithm

The ePuck embedded camera on V-rep is a vision sensor that filters the RGB colours of the blobs and other ePucks. Not collected Blobs are mapped as green and collected ones as red, and the ePuck wheels are also mapped because they have green and red parts, refer to code line 96. The data of interest that this sensor outputs are the size, centroid's 2D position, and orientation of the detected objects. Therefore, when objects are detected by the camera, a simple routine finds the biggest one which is the closest relative to the ePuck, and using (4) it can be calculated the angle of attack α_C or α_e for the blobs and ePucks respectively, refer to Fig. 2d and code line 149. The orientation value is used to differentiate between objects, for blobs is $= 0$ and for ePuck wheels is $\neq 0$, refer to code line 104.

$$\begin{aligned} \alpha_C &= \arctan\left[(x_b - x_o)/(y_b - y_o)\right] \\ \alpha_e &= \arctan\left[(x_e - x_o)/(y_e - y_o)\right] \end{aligned} \tag{4}$$

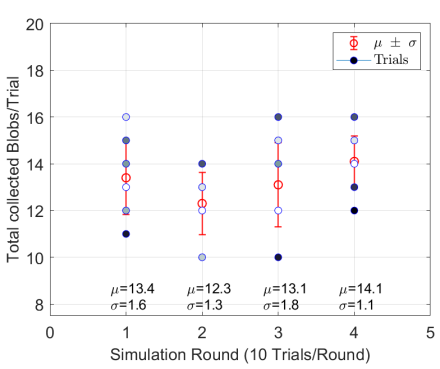
where, (x_o, y_o), (x_b, y_b), and (x_e, y_e) are the robot, blob and another ePuck wheel relative position in the 2D image. In the Random state, either the robot is foraging but does not see any blobs or is aggregating but there is no other ePuck nearby, (4) is modified with a random value w with a probability function P,

$$\begin{aligned} \alpha_{C_r} &= \alpha_C \; w \\ \alpha_{e_r} &= \alpha_e \; w \end{aligned} \tag{5}$$

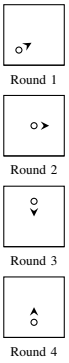
where, $P(\{w \; \epsilon \; \Omega : X(w) = 1/3\})$ and $\Omega = \{-1, 0, 1\}$, refer to code line 157 and 204.

III. RESULTS AND DISCUSSION

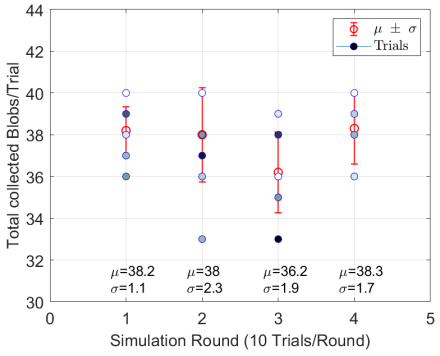
For both Scenarios, 4 Rounds of 10 trials each are simulated. Each Round has different initial positions of the robots, Fig. 3b and Fig. 3d, and each trial stops at $t = 60$. In Scenario 1, Fig. 3a shows that Round 4 has the best performance because 68% of the time the robot will forage between 13 and 15 blobs. For Scenario 2, Fig. 3b shows that Round 1 hast the best performance, 68% of the time the swarm will forage between 37 and 39 blobs. For the Aggregation case, that is simulated only in Scenario 2 Fig. 3e and Fig. 3f, Round 2 shows the best results, 68% of the time between 2 and 4 agents aggregate at some random point.



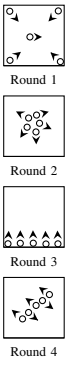
(a) Scenario 1 for $t \leq 60$ (1 robot)



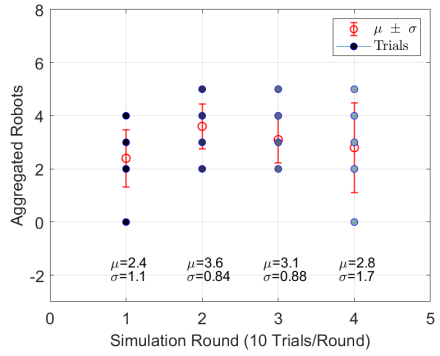
(b)



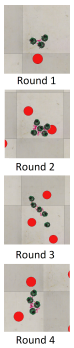
(c) Scenario 2 for $t \leq 60$ (5 robots)



(d)



(e) Scenario 2 for $60 < t \leq 120$



(f)

Fig. 3: Simulation results

References

[1] Jawhar Ghommam, Maarouf Saad, and Faical Mnif. "Formation path following control of unicycle-type mobile robots". In: *2008 IEEE International Conference on Robotics and Automation*. IEEE, 2008. DOI: 10.1109/robot.2008.4543495.

[2] I. Susnea et al. "The bubble rebound obstacle avoidance algorithm for mobile robots". In: *IEEE ICCA 2010*. IEEE, 2010. DOI: 10.1109/icca.2010.5524302.

IV. APPENDIX

```
-- The University of Sheffield
-- ACS6121 Robotics and Autonomous Systems Spring 2018/19
-- V-rep Simulation Assignment
-- R. No. : 180123717
-- Name: Paulo Roberto Loma Marconi
-----------------------------------------------------------------------------------------------------------
sim.setThreadAutomaticSwitch(false) -- manually switch the thread so we can control the sample period

-- init randomseed
math.randomseed(os.time())
math.random(); math.random(); math.random()

-- global constants
T=200 -- sample period [ms]
pi=math.pi

-- Bubble Rebound algorithm constants
N=6; alpha0=pi/N;

alphaR=0 -- [rad]
omega=0 -- [rad/s]

-- e-puck constants
-- http://www.e-puck.org/index.php?option=com_content&view=article&id=7&Itemid=9
-- http://www.gctronic.com/e-puck_spec.php
maxWheelVel=6.24 -- Max angular wheel speed 6.24[rad/s]
maxVx=0.127 -- Max robot linear velocity, 0.127[m/s]=12.7[cm/s]
L=0.051 -- 5.1 cm, distance between the wheels
D=0.041 -- 4.1 cm, wheel diameter
R=D/2 -- wheel radius

timeSimul=60 -- simulation time threshold [s]

-- Functions: ---------------------------------------------------------------------------------------------
-- Color Blob detection
function colorDetect(idx,blobPosX,blobPosY)
    local blobCol=sim.getVisionSensorImage(ePuckCam,resu[1]*blobPosX[idx],resu[2]*blobPosY[idx],1,1)
    if (blobCol[1]>blobCol[2])and(blobCol[1]>blobCol[3]) then color='R' end
    if (blobCol[2]>blobCol[1])and(blobCol[2]>blobCol[3]) then color='G' end
    if (blobCol[3]>blobCol[1])and(blobCol[3]>blobCol[2]) then color='B' end
    return color
end

-- Biggest Blob
function bigBlob(blobSize)
    local maxVal,idx=-math.huge
    for k,v in pairs(blobSize) do
        if v>maxVal then
            maxVal,idx=v,k
        end
    end
    return idx
end

-----------------------------------------------------------------------------------------------------------
-- This is the Epuck principal control script. It is threaded
threadFunction=function()
    while sim.getSimulationState()~=sim.simulation_advancing_abouttostop do
    t=sim.getSimulationTime()

-- Image Processing Part ======================================================================
    sim.handleVisionSensor(ePuckCam) -- the image processing camera is handled explicitely, since we do
    not need to execute that command at each simulation pass
    result,t0,t1=sim.readVisionSensor(ePuckCam) -- Here we read the image processing camera!
    resu=sim.getVisionSensorResolution(ePuckCam) -- Color blob detection init
```

```
65
66 -- The e-puck robot has Blob Detection filter. The code provided below get useful information
67 -- regarding blobs detected, such as amount, size, position, etc.
68
69     -- t1[1]=blob count, t1[2]=dataSizePerBlob=value count per blob=vCnt,
70     -- t1[3]=blob1 size, t1[4]=blob1 orientation,
71     -- t1[5]=blob1 position x, t[6]=blob1 position y,
72     -- t[7]=blob1 width, t[8]=blob1 height, ..., (3+vCnt+0) blob2 size,
73     -- (3+vCnt+1) blob2 orientation, etc.
74
75     pO={0.5,0} --[x0,y0] Relative Robot position in the 2D image
76     blobSize={0}; blobOrientation={0};
77     blobPos={0}; blobPosX={0}; blobPosY={0}
78     blobBoxDimensions={0};
79     blobColor={0};
80     blobRedSize={0}; blobRedPosX={0}; blobRedPosY={0};
81     blobGreenSize={0}; blobGreenPosX={0}; blobGreenPosY={0};
82     ePuckOrientation={0}; ePuckSize={0}; ePuckPos={0}; ePuckPosX={0}; ePuckPosY={0};
83
84     if (t1) then -- (if Detection is successful) in t1 we should have the blob information if the camera
       was set-up correctly
85         blobCount=t1[1]
86         dataSizePerBlob=t1[2]
87         lowestYofDetection=100
88         -- Now we go through all blobs:
89         for i=1,blobCount,1 do
90             blobSize[i]=t1[2+(i-1)*dataSizePerBlob+1]
91             blobOrientation[i]=t1[2+(i-1)*dataSizePerBlob+2]
92             blobPos[i]={t1[2+(i-1)*dataSizePerBlob+3],t1[2+(i-1)*dataSizePerBlob+4]} --[pos x,pos y]
93             blobPosX[i]=t1[2+(i-1)*dataSizePerBlob+3]
94             blobPosY[i]=t1[2+(i-1)*dataSizePerBlob+4]
95             blobBoxDimensions[i]={t1[2+(i-1)*dataSizePerBlob+5],t1[2+(i-1)*dataSizePerBlob+6]} -- [w,h]
96             -- Color detection of all blobs and group them by two vectors (Green and Red)
97             blobColor[i]=colorDetect(i,blobPosX,blobPosY)
98             if (blobColor[i]=='R') then
99                 blobRedSize[i]=blobSize[i]; blobRedPosX[i]=blobPosX[i]; blobRedPosY[i]=blobPosY[i];
100            end
101            if (blobColor[i]=='G') then
102                blobGreenSize[i]=blobSize[i]; blobGreenPosX[i]=blobPosX[i]; blobGreenPosY[i]=blobPosY[i];
103            end
104            -- Detect the orientation, size and position of the detected ePucks
105            if (blobOrientation[i]~=-0) then
106                ePuckOrientation[i]=blobOrientation[i];
107                ePuckSize[i]=blobSize[i]; ePuckPos[i]=blobPos[i];
108                ePuckPosX[i]=blobPosX[i]; ePuckPosY[i]=blobPosY[i];
109                flagEPuck=1;
110            end
111        end
112    end
113
114 -- Proximity sensor readings ================================================================================
115    s=sim.getObjectSizeFactor(bodyElements) -- make sure that if we scale the robot during simulation,
       other values are scaled too!
116    noDetectionDistance=0.05*s
117    proxSensDist={noDetectionDistance,noDetectionDistance,noDetectionDistance,noDetectionDistance,
       noDetectionDistance,noDetectionDistance,noDetectionDistance,noDetectionDistance}
118    for i=1,8,1 do
119        res,dist=sim.readProximitySensor(proxSens[i])
120        if (res>0) and (dist<noDetectionDistance) then
121            proxSensDist[i]=dist
122        end
123    end
124
125 -- Controller Algorithm ======================================================================================
126
127    -- Behaviour state: ---------------------------------------------------------------------------------------
128    if (t<=timeSimul) then behaviour='avoider'
129    else behaviour='follower' end
130
131    -- Define the weight vector
132    if (behaviour=='avoider') then
133        alpha={-3*alpha0,-2*alpha0,-1*alpha0,1*alpha0,2*alpha0,3*alpha0} -- avoider weight vector
134    elseif (behaviour=='follower') then
135        alpha={3*alpha0,2*alpha0,1*alpha0,-1*alpha0,-2*alpha0,-3*alpha0} -- follower weight vector
136    end
```

```
137
138  -- Rebound avoider/follower algorithm ------------------------------------------------------------
139    -- Calculate the angle of attack alphaR
140    sum_alphaD=0; sumD=0;
141    for j=1,N,1 do
142        sum_alphaD=sum_alphaD+alpha[j]*proxSensDist[j]
143    end
144    for j=1,N,1 do
145        sumD=sumD+proxSensDist[j]
146    end
147    alphaR=sum_alphaD/sumD
148
149    -- Foraging State: ---------------------------------------------------------------------------------
150  -- Search blobb/ePuck algorithm
151    -- Find the biggest green Blob index using the blobGreenSize vector data
152    idx=bigBlob(blobGreenSize)
153
154    -- Angle to the closest green Blob given by the biggest blob idx
155    alphaC=math.atan((blobGreenPosX[idx]-pO[1])/(blobGreenPosY[idx]-pO[2]))
156
157    -- Random State: Makes a random movement when there is no Blob detection
158    if (math.deg(alphaC)==-90) or (math.deg(alphaC)==90) then
159        alphaC=2*alphaC*math.random(-1,1)
160    end
161
162    -- Agregation State for 60<t<=120: -----------------------------------------------------------------
163    -- Find the biggest ePuck wheel
164  idxEPuck=bigBlob(ePuckSize)
165  -- Angle to the biggest ePuck wheel
166    alphaEPuck=math.atan((ePuckPosX[idxEPuck]-pO[1])/(ePuckPosY[idxEPuck]-pO[2]))
167
168 -- Ouput ==========================================================================================
169
170    -- Vx for avoider/follower -------------------------------------------------------------------------
171    threshold=0.015 -- threshold detection
172    Vx=maxVx -- go straight
173
174    if (behaviour=='avoider') then
175        if (proxSensDist[2]<threshold) or (proxSensDist[3]<threshold) or (proxSensDist[4]<threshold) or (
     proxSensDist[5]<threshold)  then
176            Vx=0; -- stop robot
177            -- Corrected angle due the symmetrical obstacle in front of the robot, only applicable in the
      avoider
178            if alphaR==0 then alphaR=pi*math.random(-1,1) end
179        end
180    end
181
182    if (behaviour=='follower') then
183        Vx=maxVx
184        if (proxSensDist[2]<threshold) or (proxSensDist[3]<threshold) or (proxSensDist[4]<threshold) or (
     proxSensDist[5]<threshold) then
185            Vx=0; -- stop robot
186        end
187    end
188
189    -- Obstacle Detection/noDetection flag -------------------------------------------------------------
190    if (proxSensDist[1]==0.05)and(proxSensDist[2]==0.05)and(proxSensDist[3]==0.05)and(proxSensDist[4]==0
     .05)and(proxSensDist[5]==0.05)and(proxSensDist[6]==0.05) then
191        flag='noObsDetection'
192    else
193        flag='ObsDetection'
194    end
195
196    -- Output omega [rad/s]=instantaneous robot angular velocity. T[ms]/1000[ms]=t[s] -------------------
197    if (t<=timeSimul) then -- avoider+ObsDetection/noObsDetection+alphaR+alphaC
198        if (flag=='ObsDetection') then
199            omega=alphaR/(T/1000); flg='Rebound';
200        elseif (flag=='noObsDetection') then
201            omega=alphaC/(T/1000); flg='Camera';
202        end
203    else                    -- follower +alphaEPuck
204        -- Random state: Random movement when there is no ePuck detection
205        if (math.deg(alphaEPuck)==-90) or (math.deg(alphaEPuck)==90) then
206            alphaEPuck=2*alphaEPuck*math.random(-1,1)
207        end
```

```
208         if (flagEPuck~=1) then
209             alphaEPuck=0;
210         end
211         omega=alphaEPuck/(T/1000); flg='ePuck';
212     end
213
214     -- Angular velocities of the wheels using the Unicycle model, vr and vl  ----------------------------
215     velLeft=(2*Vx+omega*L)/(2*R); -- rad/s
216     velRight=(2*Vx-omega*L)/(2*R); -- rad/s
217
218     -- Wheel velocity constraints ------------------------------------------------------------------------
219     if (velLeft>maxWheelVel) then velLeft=maxWheelVel
220     elseif (velLeft<-maxWheelVel) then velLeft=-maxWheelVel end
221     if (velRight>maxWheelVel) then velRight=maxWheelVel
222     elseif (velRight<-maxWheelVel) then velRight=-maxWheelVel end
223
224     -- Right/Left motor output  --------------------------------------------------------------------------
225     sim.setJointTargetVelocity(leftMotor,velLeft)
226     sim.setJointTargetVelocity(rightMotor,velRight)
227
228   print('time',t,'behaviour',behaviour,'flg',flg,'Vx',Vx,'omega',omega,'velLeft',velLeft,'velRight',
      velRight)
229
230     sim.switchThread() -- Don't waste too much time in here (simulation time will anyway only change in
      next thread switch)
231                        -- we switch the thread now!
232
233
234     end -- end while
235 end --  end thread function
236
237 ---------------------------------------------------------------------------------------------------------
238 -- These are handles, you do not need to change here. (If you need e.g. bluetooth, you can add it here)
239
240 sim.setThreadSwitchTiming(T) -- We will manually switch in the main loop (200)
241 bodyElements=sim.getObjectHandle('ePuck_bodyElements')
242 leftMotor=sim.getObjectHandle('ePuck_leftJoint')
243 rightMotor=sim.getObjectHandle('ePuck_rightJoint')
244 ePuck=sim.getObjectHandle('ePuck')
245 ePuckCam=sim.getObjectHandle('ePuck_camera')
246 ePuckBase=sim.getObjectHandle('ePuck_base')
247 ledLight=sim.getObjectHandle('ePuck_ledLight')
248
249 proxSens={-1,-1,-1,-1,-1,-1,-1,-1}
250 for i=1,8,1 do
251     proxSens[i]=sim.getObjectHandle('ePuck_proxSensor'..i)
252 end
253
254
255 res,err=xpcall(threadFunction,function(err) return debug.traceback(err) end)
256 if not res then
257     sim.addStatusbarMessage('Lua runtime error: '..err)
258 end
```