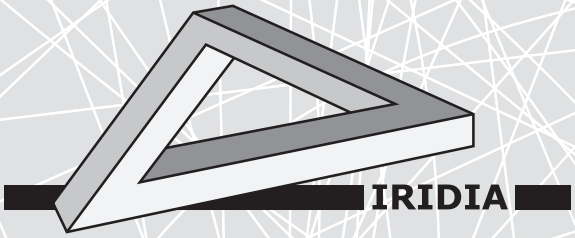

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Distributed Control to Implement Self-Assembly Strategies for the Hill Crossing Task

Rehan O'Grady, Roderich Gross,
Anders Lyhne Christensen and Marco Dorigo

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2009-022

August 2009

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:

IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2009-022

Revision history:

TR/IRIDIA/2009-008.001 August 2009

Distributed Control to Implement Self-Assembly Strategies for the Hill Crossing Task

Rehan O'Grady rogrady@ulb.ac.be
Roderich Gross roderich.gross@epfl.ch
Anders Lyhne Christensen christensen@iscte.pt
Marco Dorigo mdorigo@ulb.ac.be
IRIDIA, Université Libre de Bruxelles, Brussels, Belgium

August 2009

Abstract

This document describes the distributed control required to implement various self-assembly strategies for a hill-crossing task.

1 The *Basic Self-Assembly Response* Strategy

In this section, we describe the *basic self-assembly response* strategy and present the distributed control that enables the corresponding group level behaviour for our hill crossing task. The *basic self-assembly response* strategy is illustrated in Figure 1. The robots start by trying to execute a task independently. If they fail to complete the task independently, they self-assemble and attempt to solve the task as a larger composite robotic entity.

1.1 Strategy implementation for the hill crossing task

In our implementation, the s-bots start to navigate independently towards the target light source. In the absence of any steep hill, the robots complete the task independently. If, however, the robots detect a hill that is too steep to be navigated individually, they first aggregate, then self-assemble into a larger connected entity. The robots then navigate collectively to the target light source.

The distributed control we use to implement the strategy is shown in Figure 2. An s-bot starts by illuminating its blue LEDs and navigating independently towards the light source in the target area (state `Independent_Phototaxis`). The light source is detected with the camera. While navigating towards the light source, the s-bot uses its proximity sensors to move away from arena walls and other robots that are too close (state `Avoid_Obstacle`). If the s-bot finds itself on a hill too difficult for it to pass alone (detected using its 3D accelerometers), or if it sees (i.e., detects with its camera) a green s-bot or a red s-bot, it illuminates its green LEDs and retreats

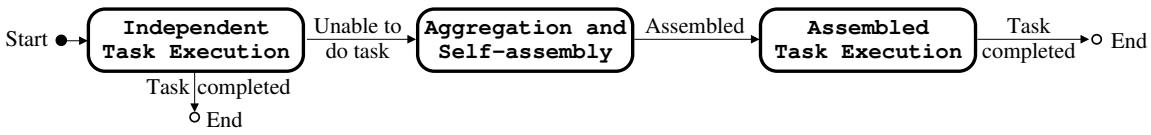

Figure 1: *Basic self-assembly response* strategy: group-level behaviour.

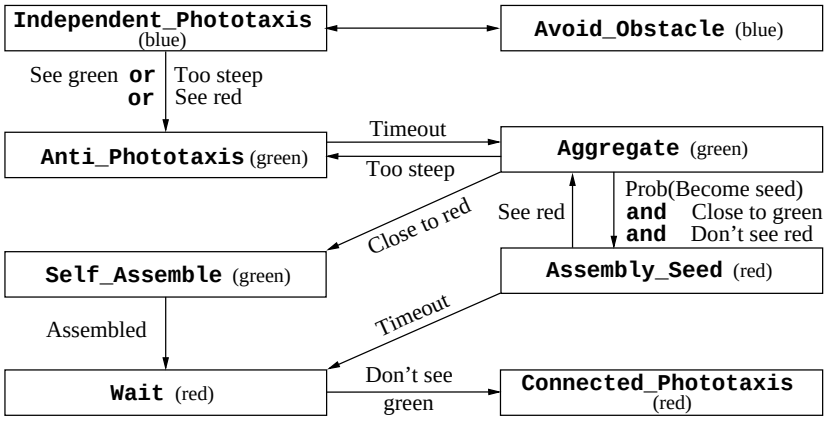

Figure 2: Distributed control to implement the *basic self-assembly response* strategy for the hill crossing task. This finite state machine controller is executed independently on individual robots. The starting state is `Independent_Phototaxis`. Colours in parentheses refer to the LEDs that are illuminated in the corresponding state.

away from the hill for a given length of time (state `Anti_Phototaxis`). It then switches into state `Aggregate`, and tries to get close to a red s-bot (the seed or an s-bot already assembled to the seed), or if no red s-bot is perceived, to search for and get close to another green (aggregating) s-bot. In the latter case, if the s-bot is sufficiently close to another green s-bot and can still see no other red s-bots, it can become, with a given probability, the seed that triggers the process of self-assembly (state `Assembly_Seed`). A seed s-bot lights up its red LEDs, and waits until a timeout has expired. If it sees another red s-bot within the timeout period, it reverts to state `Aggregate`. Otherwise, after the timeout it switches to state `Wait`. If an aggregating s-bot gets sufficiently close to a red (assembled) s-bot, then it starts self-assembling (state `Self_Assemble`). Assembled s-bots switch to state `Wait`. S-bots in state `Wait` illuminate their red LEDs. When they no longer see any green (aggregating or assembling) s-bots, they switch to state `Connected_Phototaxis` and navigate collectively to the light source. During collection navigation, each connected s-bot performs greedy phototaxis (i.e., each s-bot heads in a direct line towards the light source). After self-assembly, however, the orientations of the constituent s-bots' turrets are fixed. Each s-bot, therefore, continually rotates its turret to compensate for the track movements that move the s-bot towards the light source.

State `Aggregate`

```
 1: loop
 2:    if red close then
 3:       switch to state Self_Assemble
 4:    else if red far then
 5:       approach red
 6:    else if green close then
 7:       Prob(0.04) → switch to state Assembly_Seed
 8:    else if green far then
 9:       approach green
10:    else
11:       perform random walk
12:    end if
13: end loop
```

State `Self_Assemble`

```
 1: loop
 2:    (i1, i2) ← extract features from camera
 3:    (i3, i4) ← get proximity sensor readings
 4:    (o1, o2, o3) ← f(i1, i2, i3, i4)
 5:    if (o3 > 0.5) and (grasping requirements met)
       then
 6:       close gripper
 7:       if successfully connected then
 8:          switch to state Wait
 9:       else
10:          open gripper
11:       end if
12:    end if
13:    apply (o1, o2) to traction system
14: end loop
```

Figure 3: Control algorithms for state `Aggregate` (left) and state `Self_Assemble` (right).

Below, we detail the different states of the controller for the *basic self-assembly response* strategy (see Figure 2) and the state transition conditions.

- **State** `Independent_Phototaxis` (blue LEDs illuminated)

  This is the starting state. The s-bot uses its camera to navigate towards the target light source. The s-bot uses its accelerometers to reduce maximum track speed as a linear function of inclination. If the s-bots is in the danger of toppling over, or if the s-bot detects red or green objects (other s-bots that are aggregating, assembling or performing anti-phototaxis) in its vicinity, state `Anti_Phototaxis` is triggered. If the readings from the s-bot's 15 proximity sensors exceed a threshold, state `Avoid_Obstacle` is triggered.

- **State** `Avoid_Obstacle` (blue LEDs illuminated)

  The s-bot moves away from obstacles. Detectable obstacles include other s-bots and the arena walls. When the readings from the proximity sensors do not exceed a given threshold, state `Independent_Phototaxis` is triggered.

- **State** `Anti_Phototaxis` (green LEDs illuminated)

  This state helps aggregation and self-assembly to take place away from the hill, to prevent the hill from interfering with the self-assembly process. The s-bot performs anti-phototaxis for a set time-period at a reduced speed and then triggers state `Aggregate`. The speed and time-period were chosen by trial and error. To prevent collisions with other s-bots that are approaching the hill, the s-bot pauses if it detects an obstacle in its path (instead of performing obstacle avoidance).

- **State** `Aggregate` (green LEDs illuminated)

  If the s-bot is within 8cm of a red object (other assembled s-bot) then the s-bot triggers state `Self_Assemble`. If the closest red object is further away than 8cm, the s-bot moves towards the closest red object. If no red objects are detected and the s-bot is within 8cm of a green object (other aggregating or assembling s-bot), the s-bot then has a 0.04 probability to trigger state `Assembly_Seed` (probability is reapplied at each timestep). If no red objects are detected and the closest green object is further away than 8cm, the s-bot moves towards the closest green object. If no green or red objects are detected, the s-bot performs a random walk. Distances and probabilites used in this state were established by manual trial and error experimentation.

- **State** `Self_Assemble` (green LEDs illuminated)

  This state is implemented using a single-layer feedforward neural network which was designed by artificial evolution. Data from the s-bot camera, proximity sensors and gripper sensors are fed into the input layer of the neural network. The output layer of the neural network is then used to send commands to the s-bot wheels and gripper actuators.

- **State** `Assembly_Seed` (red LEDs illuminated)

  This state is necessary to trigger the self-assembly process. The s-bot remains stationary. If a red object is detected within 3 s of state initiation, control is passed to `Aggregate` state. (This prevents multiple seeding—if two nearby s-bots switch to state `Assembly_Seed`, both will revert to state `Aggregate`). Otherwise, control is passed to `Wait` state.

- **State** `Wait` (red LEDs illuminated)

  This state is necessary to prevent assembled s-bots from moving to the target while other s-bots are still trying to assemble. The s-bot remains stationary if it detects green objects in the vicinity (s-bots still assembling). As soon as the s-bot no longer detects other green objects, control is passed to state `Connected_Phototaxis`.

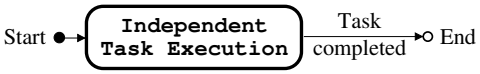

Figure 4: *Independent execution only* strategy: group-level behaviour.

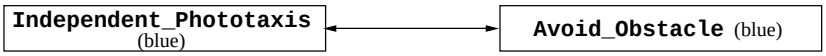

Figure 5: Distributed control to implement *independent execution only* strategy for the hill crossing task. This finite state machine is executed independently on each of the s-bots. The starting state is `Independent_Phototaxis`.

- **State** `Connected_Phototaxis` (red LEDs illuminated)

  The s-bot uses its camera to navigate towards the target light source. Because it is part of a swarm-bot, the orientation of the turret is fixed. The s-bot continually rotates the traction system with respect to the turret to keep the tracks oriented towards the target light source.

2 The *Independent Execution Only* Strategy

This strategy is illustrated in Figure 4. The distributed control to implement this strategy for the hill crossing task is a modified version of the distributed control for the *basic self-assembly response* strategy in which the transition to state `Anti_Phototaxis` is disabled (see Figure 5). Thus, only the states `Independent_Phototaxis` and `Avoid_Obstacle` are executed. The result is that each s-bot moves independently towards the light at a constant speed irrespective of the type of terrain it encounters.

3 The *Preemptive Self-Assembly* Strategy

The strategy is illustrated in Figure 6. The distributed control to implement this strategy for the hill crossing task is a modified version of the distributed control for the *basic self-assembly response* strategy in which the starting state is `Aggregate` instead of `Independent_Phototaxis` (see Figure 7). The result is that the s-bots aggregate and self-assemble irrespective of the environment. The connected swarm-bot entity then performs connected phototaxis to the light source in the target area.

4 The *Connected Coordination* Strategy

In this section, we present the more sophisticated *connected coordination* strategy that allows the assembled robots to coordinate their sensing and actuation so that they respond to their environment as a single collective robotic entity[1]. The *connected coordination* strategy is shown in Figure 8. As before, the robots self-assemble (if necessary) in response to environmental contingencies. However, the assembled robotic entity is sensitive to its collective success or failure.

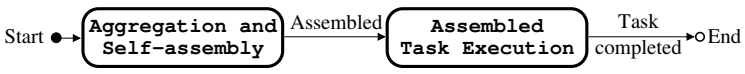

Figure 6: *Preemptive self-assembly* strategy: group-level behaviour.

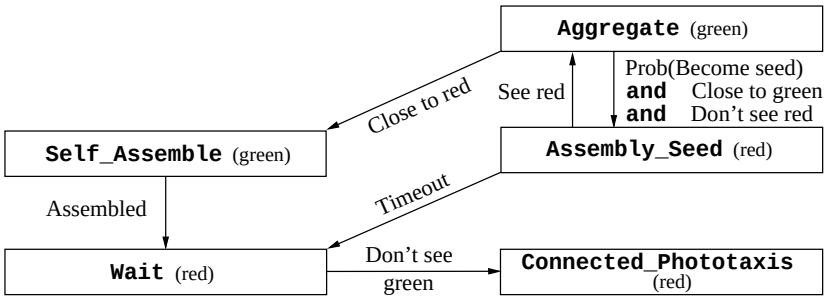

Figure 7: Distributed control to implement the *preemptive self-assembly* strategy for the hill crossing task. This finite state machine is executed independently on each of the s-bots. The starting state is `Aggregate`.

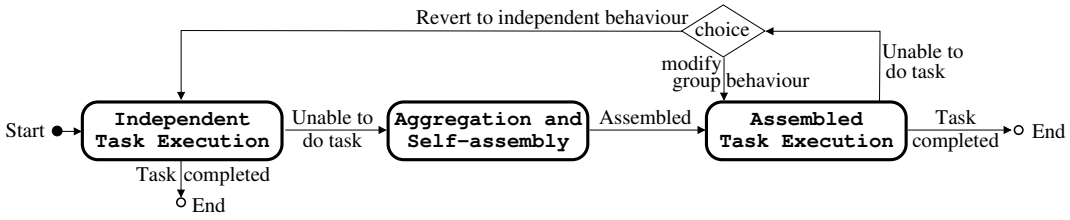

Figure 8: *Connected coordination* strategy: group-level behaviour. In this study, the choice between modifying group behaviour and reverting to independent execution is task specific and is therefore decided in advance when implementing the strategy for a particular task.

Depending on the task, if a potential group failure is detected, the assembled entity can either modify its group level behaviour or its constituent robots can revert to independent task execution.

4.1 Strategy implementation for the hill crossing task

Our implementation allows the self-assembled swarm-bot to detect when it is inappropriately oriented, and then to retreat and rotate itself to try and achieve a more appropriate orientation. The s-bots self-assemble, as before, if they encounter a difficult hill. When the self-assembled swarm-bot encounters the hill, it analyses the orientation of the hill and compares it to its own orientation. If its own orientation is insufficiently perpendicular to the orientation of the hill, the swarm-bot retreats to flat ground by performing anti-phototaxis. It then rotates until its orientation is appropriate with respect to the remembered orientation of the hill. The swarm-bot then performs phototaxis again, and upon encountering the hill again compares its orientation to that of the hill. If its orientation is appropriate, it continues to perform phototaxis and thus navigates over the hill. Otherwise, the cycle of retreating and rotating repeats until the swarm-bot has an appropriate orientation. Note that the 'choice' in this implementation of the *connected coordination* strategy is to modify the group behaviour rather than reverting to individual behaviour.

The distributed control uses a leader-follower architecture (see Figure 9). The s-bot that seeds the self-assembly process becomes the *lead s-bot* and is responsible for determining whether or not the swarm-bot is appropriately rotated. Using its LEDs, the lead s-bot issues instructions to advance, retreat or rotate to all other s-bots (*follower s-bots*) in the swarm-bot. Follower s-bots illuminate their own LEDs to mimic the LEDs of the s-bot they are gripping (which is guaranteed to be closer to the lead s-bot in a linear morphology). In this way, instructions propagate along

[1]The *basic self-assembly response* strategy did require some dedicated control to allow the individual s-bots to move while connected (see Section 1.1). However, the group level behaviour was simply what emerged from the combination of the 'greedy' behaviours of the constituent s-bots.

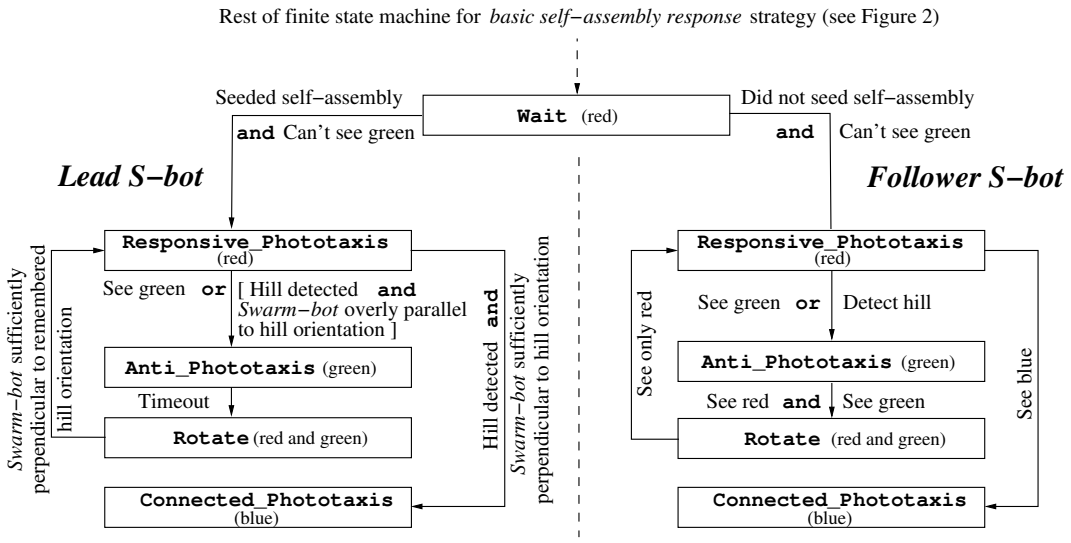

Figure 9: Distributed control to implement the *connected coordination* strategy for the hill crossing task. This distributed control consists of finite state machine extensions to the distributed control for the *basic self-assembly response* strategy. The s-bot first executes the distributed control for the *basic self-assembly response* strategy (see Figure 2). However, instead of executing the `Connected_Phototaxis` state the s-bot switches into state `Responsive_Phototaxis`, either as a lead s-bot, if it seeded self-assembly, or, otherwise, as a follower s-bot.

the swarm-bot from the lead s-bot to all the follower s-bots. The only exception is if a follower s-bot detects the hill before the lead s-bot, in which case the instruction to retreat propagates in the opposite direction.

The distributed control for the *connected coordination* strategy is an extension of the distributed control for the *basic self-assembly response* strategy. Control is branched into one of two possible finite state machine extensions (one for the lead s-bot, one for the follower s-bots) after the system has self-assembled. The two finite state machine extensions are shown in Figure 9. All of the s-bots start by executing the controller for the *basic self-assembly response* strategy, as illustrated in Figure 2. However, once they are in state `Wait`, instead of switching to state `Connected_Phototaxis`, they trigger the relevant *connected coordination* strategy finite state machine extension. In other words, the controller for the *connected coordination* strategy is constructed by substituting the state `Connected_Phototaxis` in Figure 2 with the state `Responsive_Phototaxis` in Figure 9 from either the lead s-bot finite state machine, if the s-bot seeded self-assembly, or, otherwise, the follower s-bot finite state machine.

In state `Responsive_Phototaxis`, the s-bots perform collective phototaxis while constantly checking the orientation of any hill they encounter with respect to the orientation of the swarm-bot (each s-bot uses its 3D accelerometers to determine the orientation of the hill and its camera to determine the orientation of the swarm-bot). If a hill is encountered and the orientation of the swarm-bot is appropriate with respect to the orientation of the hill (perpendicular with a tolerance of 20°), the s-bots continue performing phototaxis to the light source, but no longer check the orientation of any encountered hills (state `Connected_Phototaxis`). If the orientation of the swarm-bot is not appropriate, the s-bots remember the orientation of the hill and retreat away from the hill for a given length of time (state `Anti_Phototaxis`). They rotate until the orientation of the swarm-bot is appropriate with respect to the remembered hill orientation (state `Rotate`), and then start performing collective phototaxis again (state `Responsive_Phototaxis`).

Below, we detail the different states of the controller for the *connected coordination* strategy and the state transition conditions.

- **State** `Wait` (red LEDs illuminated)

  The s-bot remains stationary if it detects green objects in the vicinity (s-bots still assembling). As soon as the s-bot no longer detects other green objects, control is passed to state `Responsive_Phototaxis` as either a lead s-bot or a follower s-bot.

  State `Responsive_Phototaxis` (red LEDs illuminated)

  Lead S-bot

  If the lead s-bot detects a hill, it calculates whether or not the orientation of the swarm-bot is appropriate with respect to the orientation of the hill (perpendicular with a tolerance of 20°). The orientation of the hill is measured using the accelerometers. The orientation of the swarm-bot is calculated based on the positions of nearby LEDs detected with the camera. If the orientation of the swarm-bot is appropriate, the lead s-bot switches into state `Connected_Phototaxis`. Otherwise, the lead s-bot switches into state `Anti_Phototaxis`. When the lead s-bot switches into state `Anti_Phototaxis` it stores the orientation of the hill with respect to the target light source for later use in state `Rotate`.

  If the lead s-bot perceives green, it assumes that a follower s-bot of the linear swarm-bot has detected the presence of the hill, and therefore switches into state `Anti_Phototaxis`. In this case, it uses the direction towards the nearest follower s-bot (instead of its accelerometers) to roughly estimate the orientation of the hill, by assuming that the orientation of the hill is perpendicular to the orientation of the swarm-bot (this estimate is usually sufficiently accurate to ensure that the lead s-bot is the first s-bot to encounter the hill on the next approach).

  Follower S-bot

  If the follower s-bot detects a hill or perceives green, it switches into state `Anti_Phototaxis`. If the follower s-bot perceives blue, it switches into state `Connected_Phototaxis`.

- **State** `Anti_Phototaxis` (green LEDs illuminated)

  The lead s-bot switches to state `Rotate` after a timeout. Follower s-bots switch into state `Rotate` when they see red and green LEDs in front of them.

- **State** `Rotate` (red and green LEDs illuminated)

  Lead S-bot

  The lead s-bot continually compares the current orientation of the swarm-bot against the remembered (from state `Responsive_Phototaxis`) hill orientation. If the orientation of the swarm-bot is inappropriate, the swarm-bot needs to rotate—the lead s-bot moves in a direction perpendicular to the orientation of the swarm-bot so as to rotate the swarm-bot either clockwise or anti-clockwise as appropriate. When the swarm-bot is appropriately oriented with respect to the remembered hill orientation and the lead s-bot is the closest s-bot to the hill, the lead s-bot switches into state `Responsive_Phototaxis`.

  While rotating, the lead s-bot communicates rotation instructions to the follower s-bots by illuminating its green LEDs in the direction in which it is rotating and its red LEDs in the opposite direction.

Follower S-bot

If the follower s-bot can see red to its front left and green to its front right, it interprets this as an instruction to rotate the swarm-bot clockwise. It therefore moves to its left. If the follower s-bot can see red to its front right and green to its front left, it instead rotates the swarm-bot anti-clockwise by moving to its right. If the follower s-bot only detects red in front of it (without any green), it switches into state `Responsive_Phototaxis`.

- **State** `Connected_Phototaxis` (blue LEDs illuminated)

  Both the lead s-bot and the follower s-bot independently use their cameras to navigate towards the target light source. Because each s-bot is part of a swarm-bot, the orientation of the turret is fixed. Each s-bot continually rotates the traction system with respect to the turret to keep its tracks oriented towards the target light source [**?**].