Musical Abstractions in Distributed Multi-Robot Systems

Aaron Albin , Gil Weinberg , and Magnus Egerstedt

***Abstract*— In this paper, we connect local properties in a mobile planar multi-robot team to the task of creating decentralized real time algorithmic music. Using a non-linear formation control law inspired by the consensus equation, we map the local motion parameters of robots to Euclidean rhythms with the use of sequencers. The control parameters allow a human user to direct this decentralized musical process by guiding and interfering with the robots' motion, which subsequently affects their musical activity. We simulate such a robotic system in real time, demonstrating the expressiveness of the decentralized algorithmic musical output as well as a number of behaviors that arise out of the manipulation of the control parameters.**

I. Introduction

Decentralized control and coordination of multi-robot teams has arguably matured as a discipline and there is, by now, a large collection of results pertaining to a number of different applications and task domains. Examples include formation control [1], [2], [3], sensor coverage [4], and boundary protection [5], [6]. One commonality among all of these different control and coordination schemes is that they enforce a limited information flow between the robots. This limitation furthermore implies that only certain types of motions are possible and, as a consequence, only certain types of control "knobs" are available to the control designer.

In this paper we follow this line of investigation by restricting the information flow further by only allowing for locally measurable information to inform the control and coordination algorithms. Following the taxonomy from [7], by "locally measurable", we include the distance and relative orientation to adjacent robots as well as the total number of robots within sensory range.

Consider a planar robot (indexed by i) in the team, located at $x_i \in \Re^2$. If we assume that this agent has an effective footprint of Δ_i (see for example [4]), robot j is said to be adjacent to robot i if $\|x_i - x_j\| \leq \Delta_i$ and we let the number of robots adjacent to robot i be given by n_i. The locally available, measurable information is thus given by the set of distances $\|x_i - x_j\|$ and angles $\angle(x_j - x_i)$ to adjacent agents, as well as the total number n_i itself. The control laws under consideration should thus only be allowed to depend on these entities.

We couple this locally measurable information not only to mobility algorithms, but also to musical abstractions. In other words, the robots will move around in their environment and in addtiion they will make music while moving. As such, the purpose of this paper is twofold: First, we will examine how locally measurable information imposes constraints on the mobility control laws, and second, how these can mapped onto interesting musical abstractions in order to allow the robot swarm to serve as a generator of algorithmic composition.

Research on musical swarm robot systems is severely lacking. Uozumi used bug-like robots tracked with an overhead camera system [8] where the positions of the robots were given to a central computer that generated music; the robots themselves were not aware of their neighbors and did not make music. McLurkin's swarm robots were not meant to create music, but instead used to illustrate swarm behaviors [9]. In one scenario, robots would play a precomposed melody and then group together according to their MIDI[1] synthesizer voicing. Another scenario involved using the synthesizer on each robot to help with identification and debugging [10]. Both of these systems dealt only with synthetic music as opposed to physically actuated acoustic sounds.

Control mechanisms in human interactive musical robot systems, such as Pat Metheny's Orchestrion[11], allow for an instrumentalist to directly control the pitch and rhythm of the robotic system. The robots were treated like advanced physical instruments that can actuate at the level of a virtuosic jazz guitarist. In contrast to this direct approach of robot control is the marimba playing robot, Shimon[12]. An indirect method of control over Shimon exists in a "call and response module", in which the human plays a musical sequence and the robot responds to the input based on the human's style of playing or it can use internal statistical models of

Email: aalbin3@gatech.edu.
Georgia Tech Center for Music Technology, Georgia Institute of Technology, Atlanta, GA 30332, USA.
Email: gilw@gatech.edu.
Georgia Tech Center for Music Technology, Georgia Institute of Technology, Atlanta, GA 30332, USA.
Email: magnus@gatech.edu.
School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA.

[1]Musical Instrument Digital Interface

famous jazz musicians to influence and mix the musical output. Shimon was designed with the idea of being a robot musician that could play with and interact with humans socially. User studies have shown that visual contact with the robot makes for improved audience appreciation as opposed to interactions with synthetic reproductions of the same music [13]. As a physical embodied robot, it addresses the limitations of computer simulated music which cannot capture same richness as acoustic sound [14].

The novelty of this paper, from an algorithmic composition vantage point, is a new way to create interactive algorithmic music with decentralized mobile robotic systems that would be difficult to achieve with a single robot or even a group of humans. For example, consider ten mobile snare drums randomly placed in a room that obey a rule in which each drum is hit at a fixed interval only if it is within close proximity to another drum. If the drums were to converge to a single location, the resulting music would start off silent and then evolve into a repeated rhythmic pattern. While a group of experienced human drummers could carry out this task, if a conductor were to spontaneously change the rule or dictate new trajectories, then this task would become much more challenging. Although decentralized swarm music has been previously explored in interactive simulations [15], [16], they often use global parameters such as absolute position, which are typically not available to multi-robot systems, to map musical parameters as they move through a virtual space. For arbitrary locations in a room, mapping the absolute position of the robots would produce different musical output for the same type of motion; by contrast, mapping local parameters would give a musical output that is consistent with what the robots perceive about each other. Furthermore, a real musical swarm robot system would provide for a rich acoustic experience. We are interested in rhythmic creativity with multiple robot agents as this is an area largely unexplored by previous swarm based music. By understanding how to manipulate a musical swarm's motion, we can create musical behaviors that are tied directly to motion capabilities, thus providing a very expressive musical system for a human user. We first must formulate how one should control and map motion to music.

II. Algorithmic Composition

Algorithmic composition provides a strategy for creating music with swarm robots. In an analysis of a serialist musical piece by Pierre Boulez, György Ligeti provided a method by which one could analyze and also compose algorithmic compositions [17]. The composer of the work chooses what musical parameters he wishes to generate using a particular algorithmic process. For example, a random number generator may print out a set of note values. Then the composer can alter other aspects that are not governed by the process to aesthetically edit the output of the algorithm, such as by changing the loudness of each note. The result of this approach to musical composition forces the composer to let go of some aspects of the composing music, ceding some control of the creative process to the algorithm. At the same time, the composer still has some measure of creativity by shaping the often unexpected results with musical parameters which were not explicitly mapped.

We borrow from Ligeti's approach to algorithmic composition and adapt it to the scenario of real-time performance with decentralized swarm robots. In order to determine the connections between music and motion in a multi robot system, we need to examine the parameters of the nonlinear control law, choose a set of these for mappings to music, and leave the remaining parameters for human control. Since the coordinated motion of robot agents is what we wish to link to music, linking motion activity with musical activity would be a prudent choice; however, we must find an appropriate means for mapping these parameters in accordance with our musical aesthetic. One can describe musical parameters from a variety of perspectives such as pitch, frequency, amplitude, timbre, dissonance, consonance, or even emotional descriptions. Our decision to focus on rhythm with sequencers as a musical aesthetic was due to the fact that, in addition to being commonly used timing tools, they can be easily quantified and represented with arrays, manipulated with boolean functions, and can allow room for a wide range of musical expression. It is also easier to physically construct a simple percussive instrument on a small mobile robot rather than a multi-pitched instrument such as a keyboard. We will now examine the parameters of a nonlinear control law describing robot swarm motion so that we can eventually determine a mapping from local properties of the robots to musically interesting rhythms.

III. Control "Knobs"

Given a collection of N planar robots whose velocities can be directly controlled, i.e.,

$$\dot{x}_i = u_i, \ i = 1, \dots, N,$$

where $x_i \in \Re^2$ is the position of robot i. Also, assume that agent i only has access to information pertaining to agent j if $\|x_i - x_j\| \leq \Delta_i$, where Δ_i is the radius of agent i's circular, sensory footprint. We let $\mathcal{N}_i$ be the set of all neighbors to agent i, and we let n_i denote this set's cardinality.

In order to define the coordinated motions of each agent, we insist on the fact that only locally measurable information is available. Following the developments in [18], a general way of producing such a controller is to, for each $j \in \mathcal{N}_i$, define an edge-tension function $\mathcal{E}_{ij}$ as a function of $\|x_i - x_j\|$ and the desired inter-agent distance d_{ij}. Summing up all edge-tensions in the total energy in the network $\mathcal{E}$ allows us to design quite general, gradient descent-based formation controllers by letting

$$u_i = -\frac{\partial \mathcal{E}}{\partial x_i},$$

which results in $\mathcal{E}$ taking on the role of the Lyapunov function when employing LaSalle's invariance principle due to the fact that its time derivative is strictly non-increasing.

One useful aspect of this choice of control law is that it can be rewritten (see [18]) as

$$u_i = -\sum_{j \in \mathcal{N}_i} w_{ij}(\|x_i - x_j\|, d_{ij})(x_i - x_j),$$

where w_{ij} is a weight function. Note that this is in fact a weighted version of the so-called consensus equation. One additional change we can make to this equation is to add an angular offset θ_{ij} and, putting all of this together yields the following, quite general decentralized control strategy

$$\dot{x}_i = -\sum_{j \in \mathcal{N}_i} R(\theta_{ij}) w_{ij}(\|x_i - x_j\|, d_{ij})(x_i - x_j), \quad (1)$$

where R is the rotation matrix.

The general formulation in Equation 1 now tells us what the available control "knobs" really are. In fact, the items we can think of as control abstractions are the following:

- Offset angle: θ_{ij}
- Nominal interaction distance: d_{ij}
- Footprint radius (for certain classes of sensors): Δ_i

The challenge for the remainder of this paper is to connect these control abstractions to musically meaningful abstractions.

IV. Musical Abstractions

We can now turn our attention to mapping the local robot parameters to rhythm with sequencers. We will use the magnitude (speed) and angle of $\dot{x}_i$ and the degree, $\mathcal{N}_i$ (the set of all neighbors to the agent). These three parameters locally and succinctly describe how a single robot moves in relation to others while the control abstractions give us a means to influence this motion. Each robot has its own 16 step sequencer, tied to a common clock. We want increasing musical activity to correspond to filling in the steps of the sequencer; however, the rhythmic space is quantized to a set of 2^n possibilities where n is the number of steps. To make this vast rhythmic space more tractable, we use the so-called Euclid algorithm since it provides a means to create natural sounding rhythms using sequencers. As described in [19], the Euclid algorithm can generate many metrical structures found across different cultures. The function takes two arguments: the number of hits, and the total number of elements in the sequence. The hits are distributed as evenly as possible, interspersing them with empty space. Hits are represented with the symbol 1 and empty elements are represented with the symbol 0. As a simple example, $Euclid(3, 8)$ will proceed in this manner.

- 1,1,1 – 00000. Two sequences of hits are created, one with the number of hits, corresponding to the first parameter, 3, and the other with the number of empty elements, which is 8-3.
- 10, 10, 10, – 00. Empty elements are assigned to each of the hits sequentially.
- 100, 100, 10. The process continues until there are no more empty elements left to assign to the hits.
- 10010010. The subsequences are then concatenated.

In this simulation, the second parameter is fixed at 16, while the first parameter will be the result of a linear mapping f,

$$f(x) = a + (b - a)\frac{x - c}{d - c} \quad (2)$$

where x is the input value (magnitude, angle, or degree) of the robot, a and b are the minimum and maximum index of the output value respectively and c and d are the bounds of the input respectively. The result from f is truncated to an integer in order to be used in the Euclid algorithm. Changes in magnitude will correspond to changes in rhythmic activity[2]. With the angle mapping, the radian values vary from 0 to 360. Thus, as a robot rotates, it will progressively select all the possible Euclidean rhythms for ten robots. The degree of the robot agent does not require a linear mapping function since it is already an integer value whose maximum value is one less than the total number of robots[3].

[2]We impose no limits on the speed of robot agents. Thus, the maximum input value is chosen heuristically and the output value is clamped to ten, corresponding to the maximum number of robots

[3]For all mappings, we clamp the minimum output value to 1 to avoid total silence. This avoids the problem of a particular angle mapping having no rhythmic activity

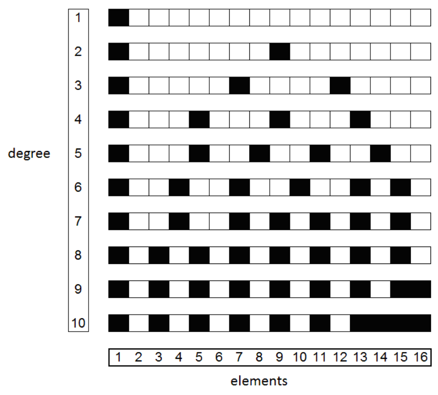

Fig. 1: A list of all possible Euclidean rhythms for ten agents.

Figure 1 shows all the possible rhythm mappings for this system with degree as the first parameter to the Euclid function and 16 as the total number of elements [4]. The time interval between each step of the sequencer is 125 ms, which corresponds to 120 beats per minute.

V. Examples

The simulation is designed in Java using the Processing API and integrated development environment [20]. Sound is produced using ten acoustic drum samples obtained from Freesound.org [21]. We use MIDI to trigger the samples and we pan the sample for each robot based on its position on the screen to help the listener discriminate among robots[5]. Robots are drawn with a circle on the screen along with their ID number and number of neighbors as shown in Figure 2. The velocity vector, $\dot{x}_i$, is represented with a blue line. A robot will flash green whenever it plays its drum sample as dictated by its sequencer. By default, the simulation starts with ten agents so that the adjacency matrix can be displayed reasonably well without obstructing the viewing area. The adjacency matrix describing the connections between agents is displayed on the top left corner of the screen. Each element ij corresponds to a directional connection from agent i to agent j.

As shown in Figure 3, the red lines indicates one way connections as described by the adjacency matrix in the upper left corner of the screen. Two way connections use black lines. The control law as described in Equation 1

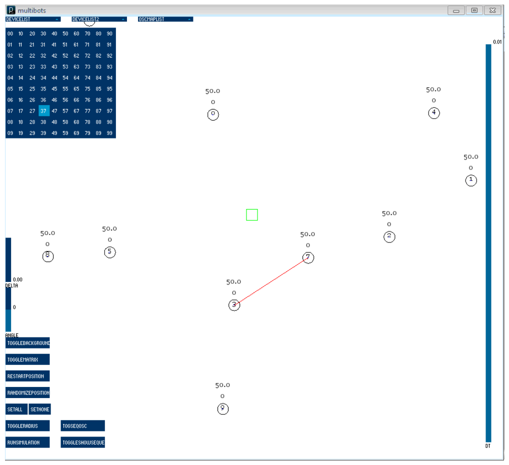

Fig. 2: A directed connection from one agent to another.

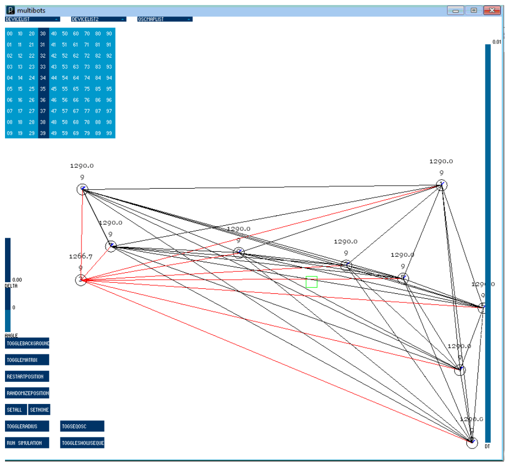

Fig. 3: Example of a robot that cannot sense other agents, with red lines indicating one way connections.

is evaluated at a time step which can be adjusted with a slider on the right side of the screen. Additional features of the program include adjustable parameters for the θ_{ij} and d_{ij} for all robots. Δ_i can also be toggled and a slider will appear to adjust this parameter.

A. *Movement to Specific Locations*

If a robot is fully disconnected and cannot sense any other robot, then it will not move; $\dot{x}_i$ is the zero vector. Figure 3 illustrates this scenario. In this situation, it makes sense for this robot to have low musical activity.

If the graph describing the adjacency matrix is not balanced, the centroid of the entire agents will shift over time as shown in Figure 4. All of the agents will move towards the agent with the zeroed out column, in this case, agent 3. The red square seen in Figure 4 indicates the initial centroid of the whole system, whereas the

[4]With 16 steps, we can make music in “common time”, that is, having four main beats per sequence, each beat having four steps. A larger step size such as 32 hits would give more options for syncopation; however, the time interval between a small number of distributed hits would be too long and the resulting rhythms would feel off-kilter.

[5]Using MIDI allows us to trigger the samples in an external synthesizer since JavaSound has audio latency issues.

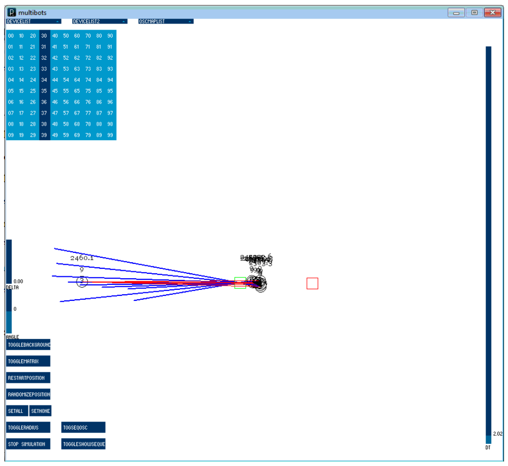
Fig. 4: The agents can sense the unconnected robot and move towards it, causing the current centroid (green square) to shift from the initial (red square). The unconnected robot does not move.

green square represents the current centroid as the agents move. Eventually, the green square will align on top of agent 3. This is an example of an unbalanced digraph.

Just as the initial starting points of the agents can be set with the mouse, they can also be modified in real time, forcing an agent to stay still. This has the same effect as in Figures 3 and 4 in a fully connected network, temporarily zeroing out one of the columns that corresponds to the stopped agent. All of the agents can be moved by a user to any arbitrary position on the screen. The user can interfere with the movement dynamics of the system momentarily and then watch as the robots adjust as a result of his or her interaction. This type of interaction behavior causes the magnitude of $\dot{x}_i$ for a robot agent to increase suddenly when there is a disturbance in the system, and then decrease as it moves towards the local centroid. In a fully connected network, all agents move to the centroid in a straight line. The program is capable of drawing traces of these pathways as shown in Figure 5. When the θ_{ij} and d_{ij} parameters of the control law are both set to zero, agents will move towards the centroid based only on distance vectors of their connected agents, slowing down as they reach the centroid. Robots that move faster will have more rhythmic density than slower moving robots when a speed mapping is used.

B. Rotational Movements

When the graph that represents the adjacency matrix is balanced, but not fully connected, agents still move towards the centroid, but they might not take a linear path. As an example, Figures 6 shows a chained two-way connection between the agents. The pathways that

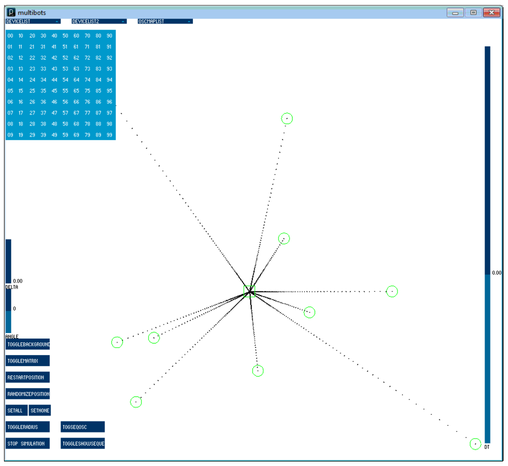
Fig. 5: An example of the path taken by a fully connected network. Robots move on a fast and direct path to centroid, slowing down as they reach the target. Speed mappings to rhythm work well iin these cases.

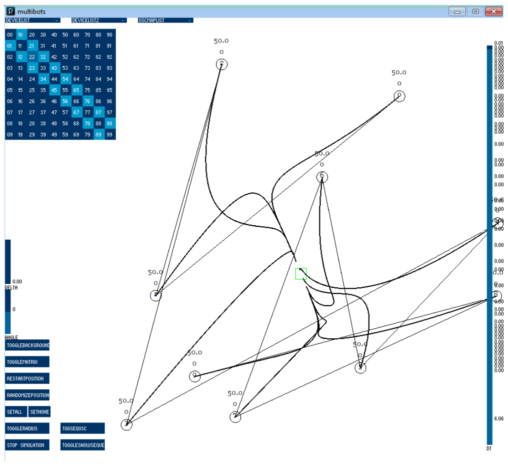
Fig. 6: Robots move to the centroid, but take a slower, curved path because they have fewer connections to others.

the agents take to the centroid are more curved in nature; it also takes longer for the agents to reach the centroid than if they were fully connected. If θ_{ij} is set, then each agent can also rotate towards the centroid at the specified angle. The amount of rotation can also be set with a slider located below the minimum distance slider on the left hand side of the screen. When θ_{ij} is less than 90 degrees in the case of a symmetric adjacency matrix (connected disoriented digraph), the agents will spiral in towards the centroid. Figure 7 shows the pathways taken by a fully connected network if the rotation parameter is set to -54 degrees (clockwise rotation).

When θ_{ij} is exactly 90 degrees, the agents will start to slightly expand further apart over time as shown in Figure 8. Incrementally, the pathways of each agent will

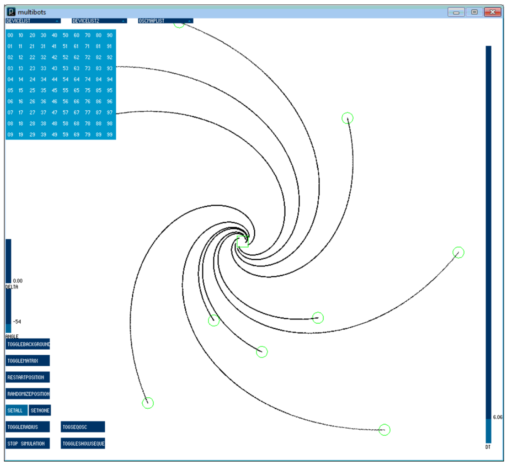
Fig. 7: An example of a fully connected network with rotation parameter at -54 degrees moving towards the centroid.

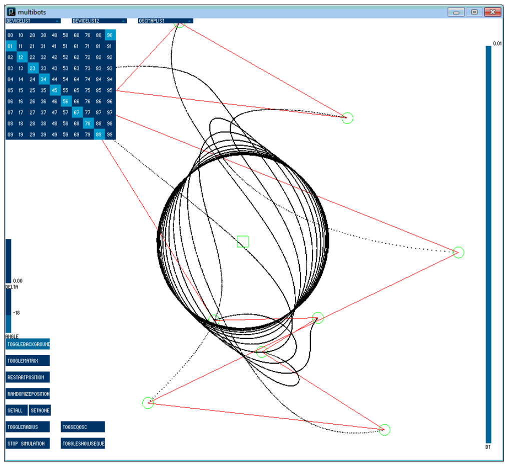
Fig. 9: A directed network orbits at fixed distance because the offset angle is π/n.

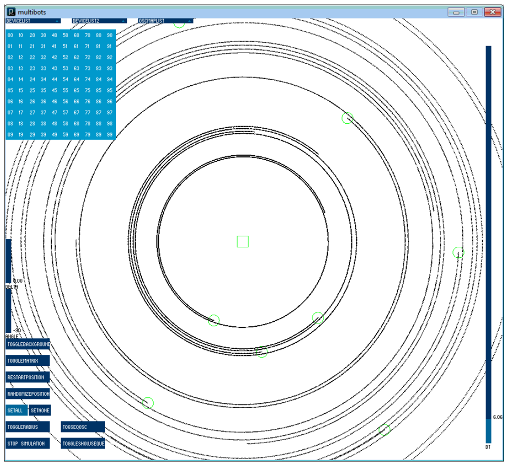
Fig. 8: Fully connected network with rotation parameter at 90 degrees. The path of the robots are slowly expanding. Angle mappings to rhythm work better in these cases.

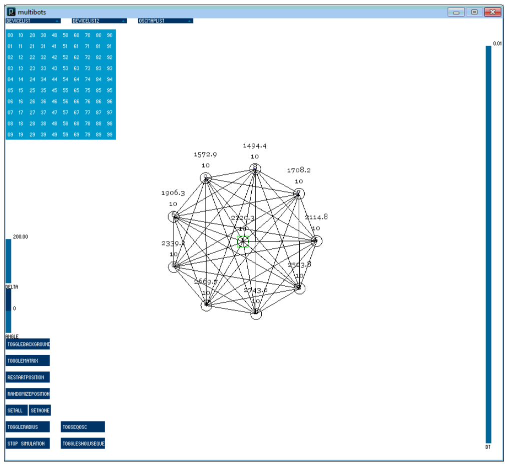
Fig. 10: In this fully connected network with d_{ij} set to 200 pixels, a formation results that can disturbed with the sliders or by the user dragging a robot and intefering with its motion. It will eventually resettle to this configuration.

get larger since at every time step the agents try to move in a tangential orbit. In the case of a weakly connected digraph with the adjacency matrix, the agents will maintain a stable rotational orbit if θ_{ij} is π/n, or in this case with ten agents, 18 degrees. Figure 9 shows the initial starting conditions of the agents, their connections, and the pathways taken that eventually lead to a circular orbit around the centroid. In these types of scenarios, the most descriptive parameter of motion would be the angle of $\dot{x}_i$. Rotation in one direction could be thought of as an increase in rhythmic density, corresponding to how the Euclidean rhythms are mapped.

In situations where the magnitude of the velocity vector is small, rotation may not be as straightforward of a mapping to use. Conversely, magnitude mappings might not seem as straightforward in perfect rotation scenarios in which the magnitude stays constant. The user is given the option of selecting the appropriate mapping with a dropdown menu bar.

C. *Maintaining Separation*

The simulation allows for modification of the interaction distance, d_{ij} with a slider bar on the left hand side of the screen. Figure 10 shows the robots settling into a formation when d_{ij} is 200 pixels. It is important to note that in this idealized view of multi-agent control, robots can only maintain a minimum distance of separation with those it can see; the simulation does not model collisions.

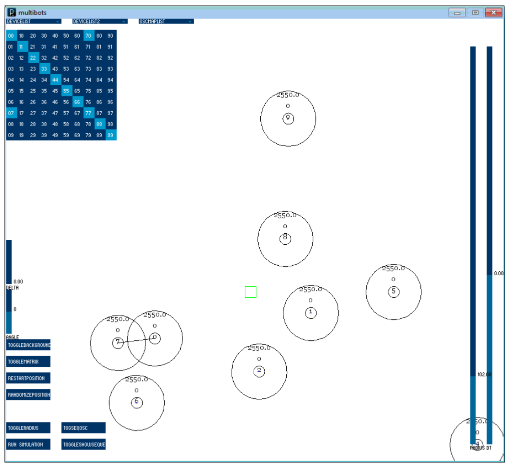

Fig. 11: An example of network with small radius for Δ_i.

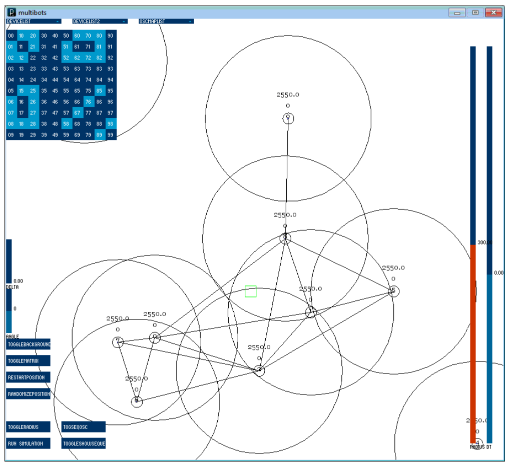

Fig. 12: When Δ_i is expanded, the adjacency matrix shows the increased number of connections. Degree mappings to rhythm work well in these cases.

D. *Footprint Based Motion*

The simulation also allows for the setting of the footprint radius, Δ_i with a slider that can be adjusted on the right hand side of the screen. To enable this, the "use radius" button must be selected so that the adjacency matrices are set by the radius as opposed to manually selecting the connections. This disk allows for the connections of the network to vary in real time. A two-way connection will form between agents if they come within range of one another.

Figure 11 shows a group of agents with Δ_i set to a small value. The resulting adjacency matrix shows only one symmetric connection between agents 0 and 7 and all other agents remain unconnected. Only agents 0 and 7 would move if the simulation were activated. As the radii are increased for all agents, as shown in Figure 12, the adjacency matrix reflects the increasing number of connections between agents. In this simulation, the adjacency matrix will always be symmetric when Δ_i is used since the radii are the same for all agents.

Using Δ_i affects both the magnitude and orientation of $\dot{x}_i$, typically by causing increased motion. Additionally, the degree of a robot becomes a dynamic parameter. A high degree value evokes a sense of increased awareness of the presence of other robot agents. Therefore, we map the increasing degree directly to the Euclid function; the more connected a robot is to its neighbors, it will be more rhythmically active. This mapping seems to work best when Δ_i is used, otherwise the rhythmic structure will not change unless the user manually changes the elements in the adjacency matrix.

VI. Discussion

In the video provided, mappings of speed can be seen from 00:00 to 01:02, angle from 01:03 to 01:50, and the remainder is degree. Several behaviors seem to arise naturally out of the use of these control knobs. One such example is a "following" behavior in which the robots are fully connected, but the user manually selects one robot and drags it to a particular location, making the other robots move. Two examples of this can be seen in the video, each using either the speed, at 00:49, or angle mapping, at 01:22. With the speed mapping, the robots are moved to a high state of rhythmic activity which then gradually subsides as the robots slow down. Angle mapping in the "following" scenario results in a strict sense of unity among the robots, as the user is essentially cycling through the Euclid rhythms by making the robots point in a particular direction. With the use of sliders for d_{ij} and θ_{ij}, the user can induce a type of "breathing" gesture among the robots. An example of this can briefly be seen from 00:34 to 00:39. Another general behavior seen throughout the video involves the user interfering with the dynamics of the control law momentarily and then letting the robots adjust. Interfering behaviors highlight the mapping from local parameters to rhythm. In a scenario with a cluster of robots using the degree mapping, robots in the center of the cluster become more rhythmically active than those on the edge. When the agents are kept separate with a d_{ij} that is near to the value of Δ_i and θ_{ij} is 90 degrees, dynamic connections in the adjacency matrix form and break apart as the robots fall in and out of one another's neighborhoods. In this behavior, the user does not need to control any knobs because the system seems to be in continual motion.

We can conclude that the control knobs of the non-linear control law give human operators the ability to create expressive rhythmic music with a decentralized multi-robot system, where the music comes about solely through local robot perspectives. This system can be directed to states of increased and decreased musical activity if we provide an appropriate mapping from the control system to the musical aesthetic. The idea of a physical swarm robot system for music becomes more compelling now that there is a methodology for creating music based on robot motion. There are some additional challenges associated with creating a real musical swarm robot system that this work does not address, such as non-holonomic motion, sources of error in sensing, state estimation, tracking and obstacle detection. Furthermore, this type of music should be evaluated by experts as to their meaningfulness and expressivity. However, assuming that the robots can reasonably determine their velocity, robot neighborhood, and if the environment is considered to be controlled and free of obstacles, we feel that a real musical swarm robot system is worth exploring. The resulting music could also be used to quickly give information to a human user about the state of the robots without the need for a visual interface.

We are developing musical robot teams using mostly inexpensive hobbyist parts. For example, a solenoid is used to strike against percussive objects such as piccolo woodblocks, paper cups, and dampened, handbells mounted on the robot. We currently use a top down camera system to track the positions of robots, but only the parameters of neighbors, angle and speed will be used to modify the robots' sequencers. Using Android phones as the brains of the robots and using Arduino-based ADK boards [22] we can design swarm robot agents capable of being controlled in a manner similar to this work and even reuse a large amount of the code in this simulation since Android runs a modified version of Java. Further work is required to take the control parameters from the control law and present them in a form more easily understood by novices or people unfamiliar with nonlinear control theory. We hope that this platform can be used to create novel ways to explore human and swarm robot interaction and interactive algorithmic music.

Acknowledgments

The work by Aaron Albin and Gil Weinberg was sponsored by the National Science Foundation. The work by Magnus Egerstedt was sponsored by the US Office for Naval Research through MURI HUNT.

References

[1] M. Ji and M. Egerstedt. Distributed Coordination Control of Multi-Agent Systems While Preserving Connectedness. *IEEE Transactions on Robotics*, Vol. 23, No. 4, pp. 693-703, Aug. 2007.

[2] W. Rei and R. Beard. Formation Feedback Control for Multiple Spacecraft via Virtual Structures. *IEE Proceedings-Control Theory and Applications*, vol. 151, no. 3, p. 357-368, May, 2004.

[3] H. Tanner, A. Jadbabaie, and G. Pappas. Flocking in Fixed and Switching Networks. *IEEE Transactions on Automatic Control*, Vol. 52, No. 5, pp. 863-868, 2007.

[4] S. Martinez, J. Cortes, and F. Bullo. Motion coordination with distributed information, *IEEE Control Systems Magazine*, Vol. 27, No. 4, pp. 75-88, 2007.

[5] J. Cortes. Distributed Kriged Kalman filter for spatial estimation, *IEEE Transactions on Automatic Control*, Vol. 54, No. 12, pp. 2816-2827, 2009.

[6] F. Zhang and N. E. Leonard. Cooperative Control and Filtering for Cooperative Exploration. *IEEE Transactions on Automatic Control*, Vol. 55, No. 3, pp. 650-663, 2010.

[7] P. Twu and M. Egerstedt. Optimal Decentralization of Multi-Agent Motions. *American Control Conference*, Baltimore, MD, July 2010.

[8] Y. Uozumi. A Musical Framework with Swarming Robots. *CMMR-LNCS*. vol. 4969. 360-67, 2008.

[9] J. McLurkin. Stupid robot tricks: A Behavior-based Distributed Algorithm Library for Programming Swarms of Robots. M.S. thesis, Dept. Elect. Eng. Comp. Sci., MIT, Cambridge, MA, 2004.

[10] J. McLurkin et al. Speaking Swarmish: Human-Robot Interface Design for Large Swarms of Autonomous Mobile Robots. /it AAAI Spring Symposium, 2006.

[11] E. Van-Burskirk. Robot Band Backs Pat Metheny on Orchestrion Tour. http://www.wired.com/underwire/2010/01/orchestrion/

[12] G. Weinberg, G. Hoffman, R. Nikolaidis, and R. Aimi. Shimon + ZOOZbeat: an improvising robot musician you can jam with. *ACM SIGGRAPH ASIA*. ACM, New York, NY, USA, 84-84, 2009.

[13] G. Hoffman, G. Weinberg. Interactive Improvisation with a Robotic Marimba Player. *Autonomous Robots*. pp 1-21, May, 2011.

[14] G. Weinberg and S. Driscol. Toward Robotic Musicianship.*Computer Music Journal*. 30, 4, pp. 28-45, 2006.

[15] T. Blackwell. Swarming and Music. *Evolutionary Computer Music*. pp. 194217, 2007.

[16] T. Unemi, D. Bisig. Playing Music by Conducting BOID Agents a Style of Interaction in the Life with A-Life. *International Conference on Artificial Life*. pp. 546 550, Boston, USA, 2004.

[17] G. Ligeti. Pierre Boulez: Decision and Automatism in Structure Ia. Die Reihe 4 (Young Composers): 3662, 1960.

[18] M. Mesbahi and M. Egerstedt. *Graph Theoretic Methods for Multiagent Networks*, Princeton University Press, Princeton, NJ, Sept. 2010.

[19] G. Toussaint. The Euclidean algorithm generates traditional musical rhythms. *BRIDGES: Mathematical Connections in Art, Music, and Science*. Banff, Alberta, Canada. July 31 to August 3, pp. 47-56. 2005.

[20] B. Fry, and C. Reas. *Processing*. http://www.processing.org/

[21] *Freesound*. http://www.freesound.org/

[22] Google, Android Open Accessory Development Kit. http://developer.android.com/guide/topics/usb/adk.html