Experimental Validation of Multi-Agent Coordination by Decentralized Estimation and Control

Michael Hwang, Matthew L. Elwin, Peng Yang, Randy A. Freeman, and Kevin M. Lynch

Northwestern Institute on Complex Systems (NICO) and
Departments of Mechanical Engineering and
Electrical Engineering and Computer Science
Northwestern University
Evanston, IL USA

Abstract. In previous work, we developed a decentralized framework for formation control by mobile robots. Each robot simultaneously estimates the current swarm shape, by local information diffusion in a communication network, while controlling its own motion to drive the swarm to the desired shape. These continuous-time estimation and control laws result in provably correct behavior, but they make unrealistic assumptions for implementation on actual robots.

This paper describes the first experimental implementation of decentralized estimation and control based on information diffusion. We develop the hardware, software, and communication protocols that adapt the theoretical approach to an actual implementation. Experiments demonstrate that the robots successfully estimate the first and second inertial moments of the swarm, using scalable local communication, while simultaneously moving to control these moments.

0.1. Introduction

We are pursuing a framework for self-organizing robot systems based on decentralized estimation and control. The goal is to "compile" desired group behaviors into local communication, estimation, and control laws running on individual robots. For properly designed control laws, the interactions of individual robots result in the desired group behavior, without any centralized control.

A key to this centralized-to-decentralized compilation process is the ability of each robot to estimate the global performance of the group based on local sensing and local communication in a time-varying communication network. For scalability, we require that the communication, computation, and storage requirements for each robot be independent of the total number of robots in the swarm. For robustness, we require that no robot be indispensable. In fact, we typically assume that each robot is identical.

To achieve robust and scalable global performance estimation, we and others have developed *dynamic average consensus estimators* [**2, 7, 5, 11, 12, 17**]. These estimators allow each robot to estimate the average of time-varying inputs from across the whole swarm. Estimates spread through the network in a process similar to heat diffusion, and we sometimes refer to the process as *information diffusion*.

Many functions of sensor inputs can be estimated using average consensus estimators, as we can first apply a transformation to the sensor inputs, pass them through the averaging process, and then apply a transformation to the output to obtain the desired result. We have shown that average consensus estimators allow the compilation of global objectives into local controllers for tasks such as formation control [**6, 16**], cooperative target tracking [**15**], estimation of the connectivity of time-varying networks [**14**], and environmental modeling by mobile sensor networks [**8**].

Until now, the study of estimation and control based on average consensus estimators has been through theory and simulation only. In this paper, we present the first experimental implementation of the approach. Our application is the formation control problem of [**6, 16**], illustrated in Figure 0.1.1. Following [**3**], we describe the desired formation of the swarm by a set of inertial moments, where the first moments describe the swarm's center of mass, the second moments describe the swarm's inertia, and third and higher moments further constrain the distribution

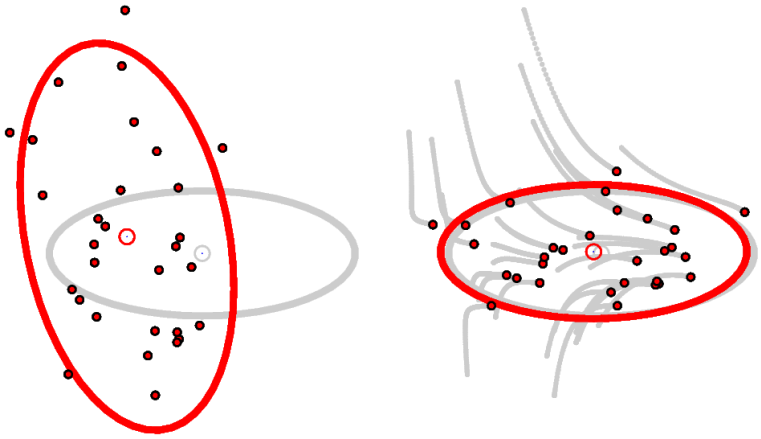

FIGURE 0.1.1. (Left) The initial configuration of a swarm, a red uniform-density ellipse with the same mass and first- and second-order moments as the swarm, and the goal formation of the swarm represented as a grey uniform-density ellipse. (Right) The swarm converges to a configuration having the desired first- and second-order moments.

of robots. The low-order moments form a convenient low-dimensional specification of the swarm shape. For example, the swarm's human supervisor could simply command the swarm to move its center of mass to a specific location, and to spread out according to some desired inertia. This frees the supervisor from having to specify a desired location for each robot.

We consider formations described by first and second moments. As shown in Figure 0.1.1, this abstraction of a planar formation can be visualized as an ellipse of uniform density with the same first moments, second moments, and mass as the swarm. By communicating with its neighbors, each robot estimates the current swarm moments and simultaneously controls its own motion to drive the estimated moments closer to the desired moments (Figure 0.1.1).

The main result of this paper is to show that the theoretical continuous-time estimation and control algorithms of [**6, 16**] can be successfully adapted to an experimental implementation of formation control. While the theoretical performance guarantees do not carry over directly to the experiment, in practice we have observed reliable convergence of our eight mobile robots to the goal formation shapes. Our experiments also suggest a number of avenues for future theoretical work.

In Section 0.2, we summarize the theoretical approach from [**6, 16**] and describe the adaptation of the communication, estimation, and control laws to the experimental implementation. In Section 0.3, we describe the "e-puck" mobile robots used in our experiments, the wireless communication system, the vision tracking system, and details of the robot control software. Section 0.4 presents the results of our experiments. Finally, Section 0.5 summarizes the results and suggests future directions.

0.2. Review of Simultaneous Estimation and Control of Formation Moments

0.2.1. Theory. Each robot is modeled as a point mass with position given by the column vector $p_i \in \mathbb{R}^m$. For our planar experiments, $m = 2$ and $p_i = [p_{i,x},\ p_{i,y}]^T$. The state of the robot is $x_i = [p_i^T,\ \dot{p}_i^T]^T$, with dynamics

$$\dot{x}_i = \begin{bmatrix} \dot{p}_i \\ u_i \end{bmatrix}, \tag{0.2.1}$$

where $u_i \in \mathbb{R}^2$ is the control force. For n robots, the configuration of the entire swarm is written $p = [p_1^T,\ \ldots,\ p_n^T]^T \in \mathbb{R}^{2n}$.

We define a goal formation for the swarm by a set of desired first- and second-order inertial moments. The first moments specify the swarm's center of mass and the second moments specify its direction-dependent spread in the plane. For a set of unit-mass point robots in the plane, the five mass-normalized first and second moments expressed in a fixed global frame can be written as the vector

$$f(p) = \frac{1}{n}\sum_{i=1}^{n} \phi(p_i) = \frac{1}{n}\sum_{i=1}^{n} [p_{i,x},\ p_{i,y},\ p_{i,x}^2,\ p_{i,x}p_{i,y},\ p_{i,y}^2]^T, \tag{0.2.2}$$

where $\phi(p)$ is called the *moment-generating function*. The desired formation of the group, $f^\star$, specifies a $(2n-5)$-dimensional submanifold of the $2n$-dimensional configuration space of the robots.

Each robot measures its own position and knows $f^\star$. To drive the robots to the goal configuration submanifold, we define a potential function

$$J(p) = [f(p) - f^\star]^T \Gamma [f(p) - f^\star] \tag{0.2.3}$$

with $\Gamma \in \mathbb{R}^{5\times 5}$ a suitably chosen symmetric positive-definite gain matrix. The control law implemented by robot i simply drives it along the negative gradient of this potential, with added damping so that the robots come to rest on the goal submanifold. The control law is

$$u_i = -B\dot{p}_i - \left[\mathscr{J}\phi(p_i)\right]^T \Gamma \left[f(p) - f^\star\right], \tag{0.2.4}$$

where $B \in \mathbb{R}^{2\times 2}$ is a positive-definite damping matrix and $\mathscr{J}\phi(\cdot)$ denotes the 5×2 Jacobian matrix of ϕ with respect to p_i.

Note that the control law (0.2.4) involves the swarm's current moments $f(p)$, a global quantity. In a decentralized implementation, each robot must replace $f(p)$ with a local estimate obtained through communication with neighboring robots. For scalability, this local communication cannot simply relay all accumulated sensory data, but rather must summarize the data so that the size of the communication packets and the data each robot must store is independent of the number of robots in the swarm.

To obtain local estimates of the global swarm moments, each robot implements a *proportional-integral (PI) average consensus estimator*. Each robot continuously receives its neighbors' estimates of the swarm's moments, "averages" them with its own, and broadcasts its new estimate. Inputs to the averaging process are the robots' measurements of their own positions. More specifically, robot i broadcasts to its neighbors its estimator output v_i and its estimator internal state w_i, and

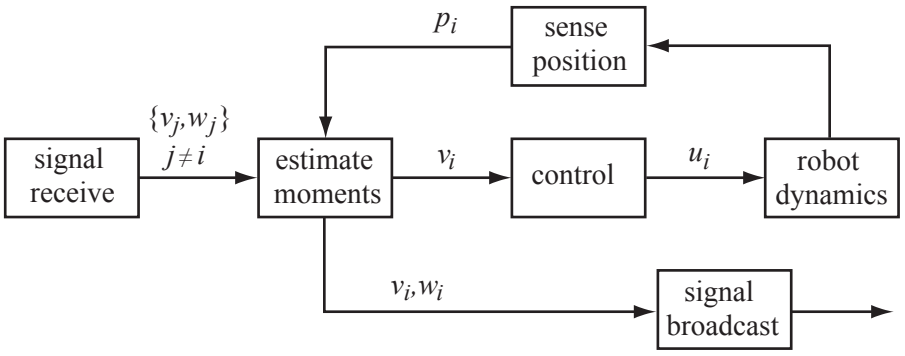

FIGURE 0.2.1. Block diagram for robot i.

calculates

$$\dot{v}_i = \gamma(\phi(p_i) - v_i) - \sum_{j \neq i} a(p_i, p_j)\big[v_i - v_j\big] + \sum_{j \neq i} b(p_i, p_j)\big[w_i - w_j\big] \tag{0.2.5}$$

$$\dot{w}_i = -\sum_{j \neq i} b(p_i, p_j)\big[v_i - v_j\big], \tag{0.2.6}$$

where $a(p_i, p_j) \geq 0$ is a proportional gain between robots i and j, $b(p_i, p_j) \geq 0$ is an integral gain between robots i and j, and $\gamma > 0$ is a global "forgetting factor." The gains $a(\cdot, \cdot)$ and $b(\cdot, \cdot)$ implement an undirected graph. When robots i and j cannot communicate, $a(p_i, p_j) = 0$ and $b(p_i, p_j) = 0$. New information enters the averaging process through the term $\gamma\phi(p_i)$ in equation (0.2.5), with γ controlling the rate at which new information replaces old information.

In [**6, 16**], we demonstrated that the PI average consensus estimator in (0.2.5) and (0.2.6) results in estimates v_i that track $f(p)$ closely, provided that the communication network is connected and $f(p)$ is changing slowly. Furthermore, if the inputs stop changing (p is constant), the estimates converge to $f(p)$ with zero steady-state error. This behavior occurs even after dropped packets, noisy messages, and robots entering or leaving the network.

Substituting the estimator output v_i for the actual moments $f(p)$ in (0.2.4) yields the decentralized control law for robot i

$$u_i = -B\dot{p}_i - \big[\mathscr{J}\phi(p_i)\big]^T \Gamma\big[v_i - f^\star\big]. \tag{0.2.7}$$

As demonstrated in [**16**], a slight modification of this control law, used in concert with the PI estimator, will cause the swarm to converge to the desired configuration submanifold from nearly every initial state, and all equilibria away from the goal submanifold are unstable.

A block diagram of robot i's control system is given in Figure 0.2.1.

0.2.2. Modifications for Experiments. To implement the theoretical estimation and control procedure described above on actual robots, we made three primary modifications:

(1) The control law (0.2.7) was modified for kinematic robots with velocities as controls, instead of point masses with forces as controls.

(2) The definition of second moments in (0.2.2), referenced to the origin, was modified to central second moments, referenced to the swarm's center of mass.
(3) The continuous-time algorithms were adapted to the experimental realities of limited communication bandwidth and digital computing, such as dropped packets and asynchronous discrete-time sensing and control.

The first two modifications are easily handled theoretically and have little impact on performance guarantees. Understanding the effects of the third modification defines an entire research agenda which we are only beginning to explore. The three modifications are discussed below.

Kinematic robots. The mobile robots, discussed in Section 0.3, are most naturally modeled as kinematic robots with velocities as controls. Therefore the state of the system is simply the robot's configuration, $x_i = p_i$, and the dynamics are written $\dot{x}_i = u_i$. With these dynamics, we simplify the gradient control law (0.2.7) to

$$u_i' = -\left[\mathscr{J}\phi(p_i)\right]^T \Gamma \left[v_i - f^\star\right]. \tag{0.2.8}$$

As shown in [**13**], kinematic robots under the control law (0.2.8) enjoy performance guarantees similar to those of point mass robots with force inputs under the control law (0.2.7).

In practice, the velocities calculated by (0.2.8) are passed through a collision-avoidance filter to prevent robot collisions, and then through a saturation stage that limits the commanded robot speed. Collision avoidance is described in more detail in Section 0.3.4.2.

Central moments. In our previous formula (0.2.2) for calculating the moments, the moment-generating function $\phi(p_i)$ was defined as

$$\phi(p_i) = [p_{i,x},\ p_{i,y},\ p_{i,x}^2,\ p_{i,x}p_{i,y},\ p_{i,y}^2]^T.$$

This choice has the disadvantage that as the swarm translates without changing shape, the second moments change. These global second moments are with respect to the origin, not the center of mass of the swarm. The result is that the behavior of the system is not translation-invariant—a set of control gains Γ that yields good behavior for one goal center of mass may yield slow convergence or undesirable oscillation for another goal center of mass. To avoid this problem, we redefine robot i's moment-generating function to be

$$\phi(p_i) = [p_{i,x},\ p_{i,y},\ (p_{i,x} - v_{i,x})^2,\ (p_{i,x} - v_{i,x})(p_{i,y} - v_{i,y}),\ (p_{i,y} - v_{i,y})^2]^T,$$

where $(v_{i,x}, v_{i,y})$ is the robot's estimate of the group's center of mass. With this choice of $\phi(p_i)$ in (0.2.6), the robots estimate the swarm's second moments with respect to its center of mass (central second moments) rather than with respect to the origin.

Asynchronous discrete-time implementation. Each robot updates its velocity according to the control law (0.2.8) every T_1 seconds, and every T_2 seconds it updates its estimate of swarm moments and broadcasts these estimates. The

discrete-time estimator is written

$$v_i = v_i^- + \gamma(\phi(p_i) - v_i^-) - K_P \sum_{j \in \mathcal{N}(i)} \left(v_i^- - v_j\right) + K_I \sum_{j \in \mathcal{N}(i)} \left(w_i^- - w_j\right) \tag{0.2.9}$$

$$w_i = w_i^- - K_I \sum_{j \in \mathcal{N}(i)} \left(v_i^- - v_j\right), \tag{0.2.10}$$

where v_i is the new swarm moments estimate, w_i is the new estimator internal state, (v_i^-, w_i^-) is the previous estimator state, p_i is the current robot position, (v_j, w_j) is the most recently received estimator state from robot j, $\mathcal{N}(i)$ is the set of robots that robot i has heard from since the last estimator update, and $K_P, K_I, \gamma > 0$ are estimator gains. Because there is no global synchronization of the robots' clocks, each robot runs its control, estimation, and communication processes on a different schedule. After updating its estimate, the robot discards estimator state information received from other robots, so that each estimator update depends only on communication received during the past T_2 seconds.

In our implementation, $T_1 = 0.2$ s and $T_2 = 0.4$ s.

0.3. Details of the Experimental Implementation

Our experimental implementation consists of eight small mobile robots moving on a 300 cm × 300 cm section of a smooth, white floor. To obtain position estimates of each robot, four cameras view the workspace from above. The top of each robot has a unique black-and-white pattern, allowing measurement of the position and orientation of each robot. Each robot is equipped with an XBee radio [**4**], and the vision system periodically broadcasts position updates to each robot's XBee. Thus the vision system and wireless broadcasts act as a GPS. Robots broadcast state estimates to each other through the same XBee radios. A data logger PC listens to the messages being broadcast by the robots to monitor the behavior of the system. The robots' estimation and control is truly decentralized—each robot runs the same software, and the communication, computation, and storage requirements are independent of the number of robots in the swarm.

An overview of the system is shown in Figures 0.3.1 and 0.3.2. More details about the mobile robot, wireless communication, indoor positioning system, robot firmware, and command system are given below.

0.3.1. Mobile Robot. For our experiments, we selected e-puck mobile robots, developed at the École Polytechnique Federale de Lausanne (Figure 0.3.3) [**9**]. We chose these robots because of their simple drive system, small size (70 mm diameter), sufficient processing power, extensibility, and relatively low cost. The two wheels of the differential-drive robot are actuated by stepper motors, allowing for accurate velocity control and dead reckoning without the need for encoders. A 16-bit Microchip dsPIC30F6014A controls the robot's motors and peripherals, and has enough processing power for our control, estimation, and communication needs.

The e-puck comes standard with Bluetooth wireless communication, which requires a "master" device to coordinate several "slave" devices. This architecture is unsuitable for decentralized estimation and control. We therefore added a Digi XBee RF module [**4**] for inter-robot communication. The e-puck's PIC microcontroller communicates with the XBee using RS-232. The wireless communication system is discussed further in Section 0.3.2.

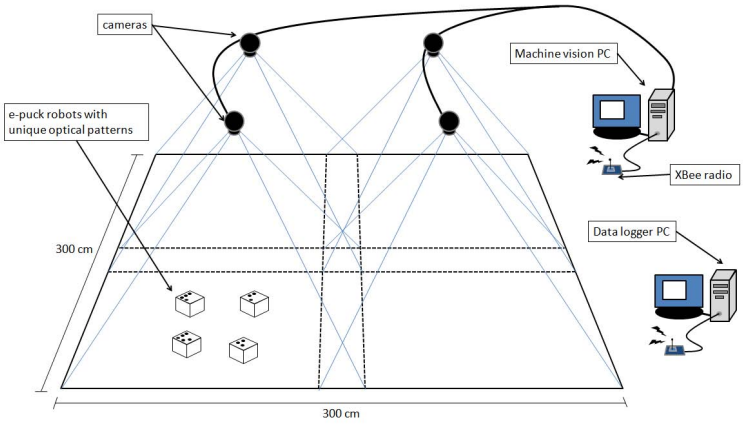

FIGURE 0.3.1. Schematic of the experimental setup.

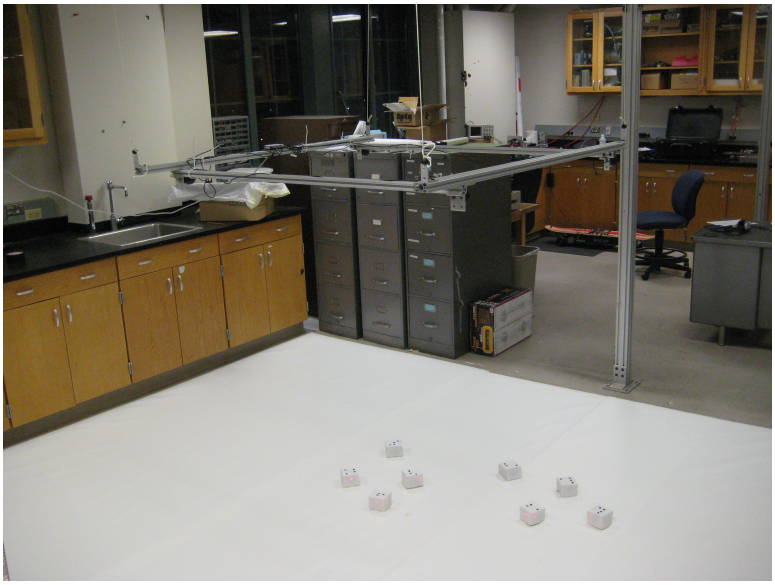

FIGURE 0.3.2. A photograph of the experimental arena.

The chassis of each robot has three degrees of freedom in the plane, represented by a point (x, y) halfway between the wheels and an angle θ corresponding to the "forward" motion direction. To recover the point robot model, we control the motion of a reference point (x_r, y_r) fixed in the robot's frame, defined as

$$(x_r, y_r) = (x + h\cos\theta, y + h\sin\theta), \quad h > 0,$$

and leave θ uncontrolled (Figure 0.3.4). For wheels of radius R separated by a distance L along the common axle line, the relationship between the commanded

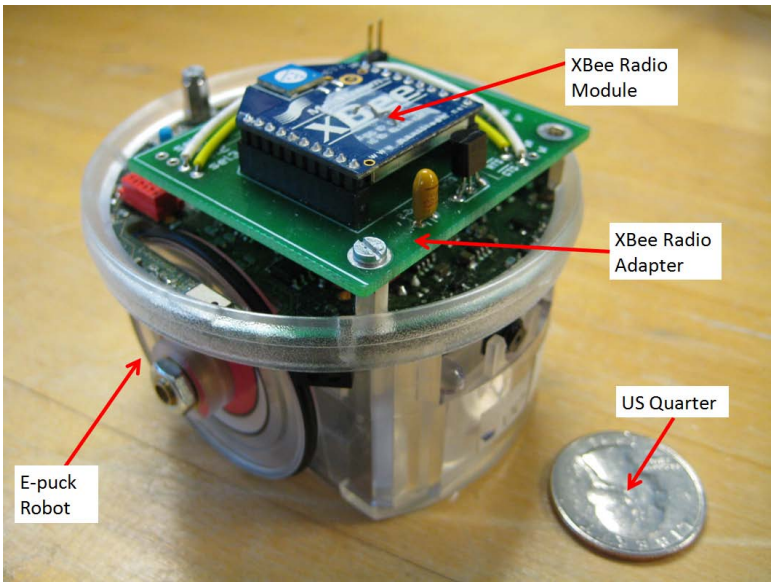

FIGURE 0.3.3. The EPFL e-puck mobile robot, with its original extension module replaced with the custom XBee radio extension module.

velocity $[u_x, u_y]^T$ of the reference point and the commanded angular velocities u_ℓ and u_r of the left and right wheels, where $u_\ell, u_\mathrm{r} > 0$ causes the robot to go forward, is

$$\begin{bmatrix} u_\ell \\ u_\mathrm{r} \end{bmatrix} = \frac{1}{2hR} \begin{bmatrix} 2h\cos\theta + L\sin\theta & 2h\sin\theta - L\cos\theta \\ 2h\cos\theta - L\sin\theta & 2h\sin\theta + L\cos\theta \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix}. \tag{0.3.1}$$

By choosing $h = L/2$, the velocity limits on the wheel speeds, represented as a square in the (u_ℓ, u_r) space, become a square in the (u_x, u_y) space of velocities of the reference point. For the e-puck, $L = 53$ mm and $R = 20.5$ mm, and we use $h = 35$ mm. The maximum wheel speed for the e-puck is 1 rev/s, yielding a maximum forward speed of approximately 130 mm/s.

0.3.2. Wireless Communication. As mentioned earlier, the robots are outfitted with XBee radios, which they use for broadcasting and receiving messages. A data logger PC also uses an XBee radio to capture the broadcast traffic. The data logger attaches timestamps to each received message so that the behavior of the system can be analyzed and plotted later. Finally, the vision PC, which is connected to the four cameras, uses an XBee radio to broadcast the positions and orientations of each of the robots.

XBee radios are low-cost, low-power (1mW) radios that use the IEEE 802.15.4 standard and operate in the 2.4 GHz ISM band. Each XBee module contains an RF transceiver and a microcontroller implementing basic networking capabilities such as packet addressing and checksums. Together, these radios form a peer-to-peer network in which each member can broadcast a message to all other members. Each XBee embeds a unique identification number in the packets it sends, allowing packet recipients to identify packet senders.

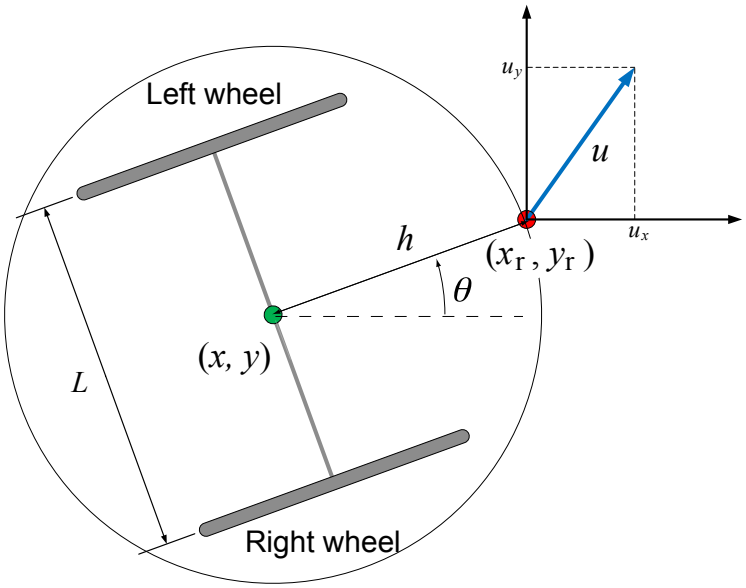

FIGURE 0.3.4. The reference point (x_r, y_r) is commanded to move with a velocity $u = [u_x, u_y]^T$, which is converted to wheel speeds in (0.3.1).

The XBee connects to the robot's microcontroller via the e-puck's extension connector. The e-puck's microcontroller communicates with the XBee module over a serial port using the RS-232 protocol at 115200 baud. Received data can be stored in a 100 byte buffer on the radio, so the microcontroller need not process received data immediately. Each packet is approximately 40 bytes long, however, so the microprocessor must process XBee data regularly to prevent buffer overflows.

The practical range of the XBee radios is on the order of tens of meters, so each robot in the test arena can hear from every other robot. To simulate radius-limited communication, each robot broadcasts its position along with its estimator state. Receiving robots simply ignore messages received from robots beyond a certain distance.

In our implementation, each robot broadcasts its estimator state every $T_2 = 0.4$ s on a randomly initialized clock. We did not implement any protocols to adapt broadcast frequency or timing based on network traffic. The vision PC also periodically broadcasts the position and orientation of each robot. Based on results from the data logger PC, we estimate that each XBee radio drops 5-10 percent of the packets that are broadcast. These dropped packets mean that the communication network may temporarily become directed (one-way communication between robots) rather than undirected (if robot i hears from robot j, then robot j hears from robot i), as assumed in the theory. This manifests itself as noise in the consensus algorithms.

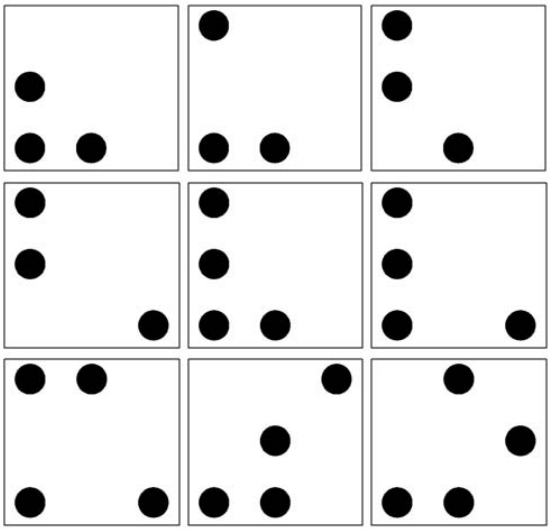

FIGURE 0.3.5. A sample of the patterns used with the vision algorithm.

0.3.3. Indoor Positioning System. To simulate GPS for a small-scale indoor environment, we developed a positioning system using webcams and OpenCV, an open-source computer vision library. Our system allows us to track the positions of multiple robots with less than 1 cm error.

The hardware for the positioning system consists of four Logitech Quickcam Communicate Deluxe USB web cameras mounted on the corners of a rectangular frame suspended above the test arena. Using multiple cameras increases the coverage of the system without the need for a single high-resolution camera and wide-angle lens. The cameras have overlapping fields of view in the interior of the arena to ensure that at least one camera images every target as it moves between fields of view.

To track the robots' positions and orientations, each robot has a unique rotationally-asymmetric pattern of black dots on a white background mounted on its top side (see Figure 0.3.5). To calibrate the cameras, a grid of 25 point markers is evenly spaced over the test arena, with each marker at the height of the robots' dot patterns. The grid is chosen so that each camera sees nine of the markers. A matrix transforming the camera image coordinate system to the real-world coordinate system is computed using the direct linear transformation given in [**1**].

The vision tracking process starts with a 640×480 pixel image from each camera. In the first step, each image is thresholded to obtain a binary image. Then the black dots are partitioned into clusters, and each robot is uniquely identified by comparing a vector of rotationally-invariant measures to known profiles associated with each robot: the number of dots in the cluster and the vector of distances from the dots to the center of mass of the dots. Once the identity of each robot is determined, its position and orientation are calculated using the positions of the dots relative to the center of mass.

All four images are collected and processed into identifiers and positions and orientations for each robot approximately 5-10 times per second. This data is displayed in real time on the vision PC monitor. The vision PC broadcasts all of the robots' positions and orientations approximately once every three seconds. In between virtual GPS updates, the robots use dead reckoning to update their position estimates. The relatively low virtual GPS update rate was chosen to reduce network traffic, and because dead reckoning is accurate over short durations.

0.3.4. Robot Firmware. The firmware code for the PIC microcontroller on the e-puck implements the distributed consensus estimator and the motion controller as described in Section 0.2.2. The code is written in C and compiled with Microchip's C30 compiler. Each robot has identical firmware, but can be differentiated by the unique identifier of its XBee radio.

When implementing these algorithms, practical limitations of the hardware, such as the microcontroller's relatively slow floating point operation performance and limited memory, must be considered. Also, the low-power XBee radio network has limited data throughput; if too many robots attempt to transmit data simultaneously, many data packets will be lost.

To account for these limitations, we limit the rate of calculating control and estimation laws and transmitting data. Every 200 ms, a timer interrupt generates a clock "tick." Every tick, the motion controller evaluates the control law (0.2.8) and updates the wheel speeds. Every second tick, the consensus estimator is evaluated and the results broadcast. This timing was chosen experimentally to yield good performance while giving the microcontroller ample time to complete its tasks. A summary of the control flow is shown in Figure 0.3.6.

Further firmware details are given below.

0.3.4.1. *Initializing the Consensus Estimator.* When a robot is turned on and first joins the communication network, its estimator state variables are initialized to zero. Any choice would suffice, as initial estimator values quickly decay.

0.3.4.2. *Collision Avoidance.* The reference point velocity u' produced by the control law (0.2.8) is passed through a simple collision-avoidance filter to produce a modified control u that prevents collisions between robots. Based on the position information broadcast by other robots, each robot determines the position of its nearest neighbor. Let $d \in \mathbb{R}^2$ be the vector from the robot to its nearest neighbor, and define $\eta = -\|u'\|d/\|d\|$ to be a "collision-avoidance velocity" in the direction of $-d$ with magnitude $\|u'\|$. We define three zones around each robot, represented by an "ignore radius" R_2 and a "proximity radius" R_1, where $R_2 > R_1$. If $\|d\| > R_2$, then there is no danger of collision and $u = u'$. If $\|d\| \leq R_1$, then the robot follows the collision-avoidance velocity ($u = \eta$). At intermediate distances, the velocity u is a convex combination of the originally calculated velocity u' and the collision-avoidance velocity η, according to the formula

$$\alpha = \frac{\|d\| - R_1}{R_2 - R_1} \tag{0.3.2}$$

$$u = \begin{cases} u' & R_2 < \|d\| \\ \alpha u' + (1-\alpha)\eta & R_1 < \|d\| \leq R_2 \\ \eta & \|d\| \leq R_1. \end{cases} \tag{0.3.3}$$

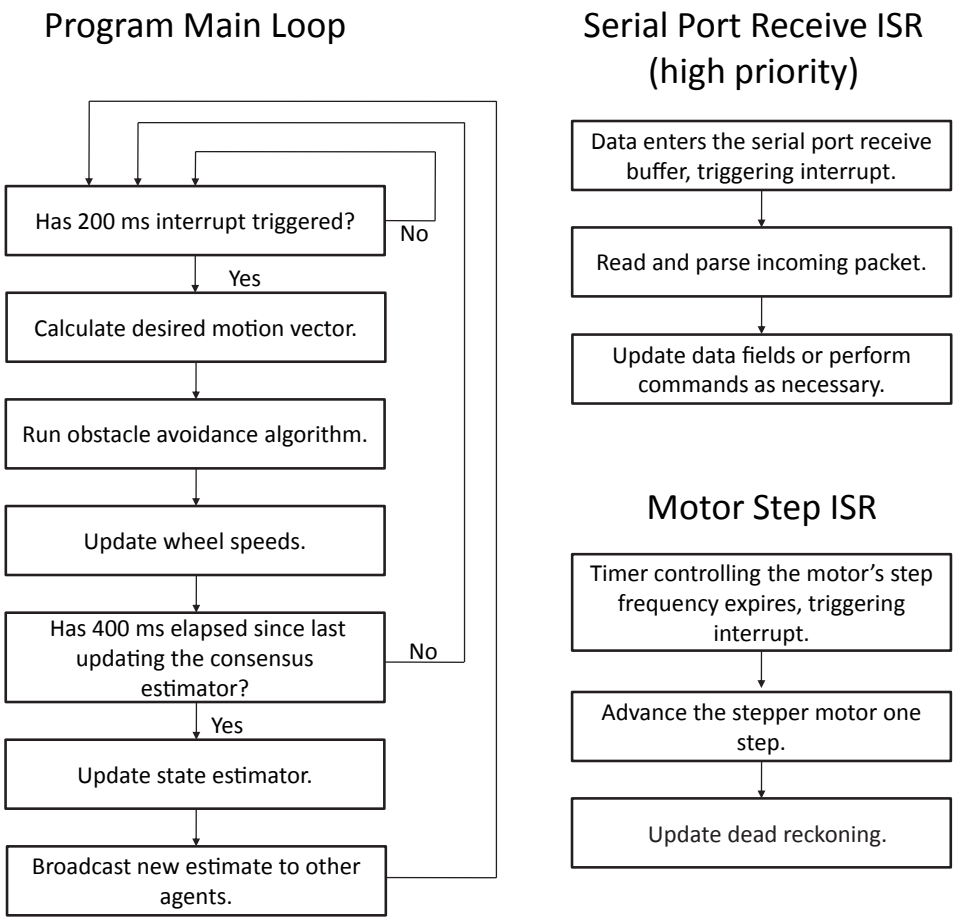

FIGURE 0.3.6. State diagram for the firmware on the e-puck robot. The high priority interrupt service routine triggered by the UART receive buffer interrupts the other interrupt service routines.

0.3.4.3. *Wheel Speed Controller and Saturation.* Every $T_1 = 0.2$ s, the robot's commanded velocity u is calculated from the control law (0.2.8) and passed through the collision-avoidance filter (0.3.3). This velocity is then converted to commanded wheel speeds using (0.3.1). If either wheel speed exceeds the maximum of 1 rev/s, both wheel speeds are scaled to maintain their ratio while satisfying the maximum speed constraint. For typical gain values Γ in our experiments, a wheel speed is saturated until the swarm is close to satisfying the desired formation moments.

While the wheel speeds are constant between evaluations of the control law, the linear velocity of the reference point is generally not constant, due to the changing angle of the robot. This deviation from the theory does not cause significant practical issues.

The microcontroller implements a commanded wheel speed by converting the speed to a time between wheel increments and setting a timer to interrupt with

this period. The timer's interrupt service routine increments the wheel angle and updates the dead reckoning estimate.

0.3.5. Command System. The vision PC doubles as a command console for the swarm, allowing the broadcast of control parameter updates, changes in goal behavior, and other user interaction. Example commands include

- sleep (put one or more robots into a inactive state)
- wake (put one or more robots into an active state)
- change goal state ($f^\star$)
- change consensus estimator gains (K_P, K_I, γ)
- change motion controller gains (Γ)
- change maximum communication radius

0.4. Experimental Results

We tested the swarm under several scenarios:

(1) Moving to a goal formation.
(2) Repeating the first experiment using poorly chosen motion control gains.
(3) Repeating the first experiment when the robots have a limited communication radius.
(4) Moving the swarm through a bottleneck by giving successive goal formations.

Each experiment used eight e-puck robots. As the experiments ran, the data logging PC recorded all of the packets broadcast. The plots in this section use data extracted from these packets. In several of the following figures, data logged from the experiment is plotted over a frame from a video recording of the experiment, taken by a dedicated video camera (not one of the vision system webcams). The green ellipse is a graphical representation of the swarm's actual moments, and the red ellipse represents the goal moments. The moment estimates of each robot are also plotted against time, along with the goal moments (blue dashed line) and the actual moments of the swarm (red dashed line).

In these experiments, the major source of noise was packet loss. When a packet from a robot is lost, it appears as if the robot were momentarily removed from the swarm, so the rest of the swarm compensates by shifting position, causing jittering when the swarm is near its goal state. This jittering is reduced, at the expense of some steady-state error, by imposing a deadband on the control u: if the norm of u is less than some threshold, the robot remains stationary.

We chose controller and estimator gains experimentally by observing their effect on the swarm. Unless otherwise specified, we used the following parameters, where $(0, 0)$ is the center of the arena:

- Goal: $f^\star = [100 \text{ mm}, 300 \text{ mm}, 160000 \text{ mm}^2, 40000 \text{ mm}^2, 40000 \text{ mm}^2]^T$. In the plots of the experimental results, first and second moments are referred to as $[x_{\text{cm}}, y_{\text{cm}}, I_{yy}, I_{xy}, I_{xx}]^T$.
- Motion control gains: $\Gamma = \text{diag}(1 \text{ s}^{-1}, 1 \text{ s}^{-1}, 1 \times 10^{-5} \text{ mm}^{-2}\text{s}^{-1}, 2\times10^{-5} \text{ mm}^{-2}\text{s}^{-1}, 1\times10^{-5} \text{ mm}^{-2}\text{s}^{-1})$.
- Collision avoidance parameters: $R_2 = 200$ mm, $R_1 = 100$ mm.
- Communication radius: infinite.
- Estimator gains for robot i: $K_P = 0.7/|\mathcal{N}(i)|$, $K_I = 0.1/|\mathcal{N}(i)|$, $\gamma = 0.05$, where $|\mathcal{N}(i)|$ is the number of robots that robot i has received information

from in the past T_2 seconds. (For a known bound on the number of robots, it may be better to choose K_P and K_I as constants, to avoid the possibility of introducing unbalanced networks when $|\mathcal{N}(i)|$ is different for each robot; see [**5**].)

0.4.1. Simple Formation Control. In this experiment, the robots are scattered throughout the test arena and the goal is fixed. Figure 0.4.1 shows the starting positions of the robots and the trace of their paths as they converge to the desired configuration manifold. Figure 0.4.2 plots the first and second moment estimates of each of the robots, as well as the actual moments, as a function of time.

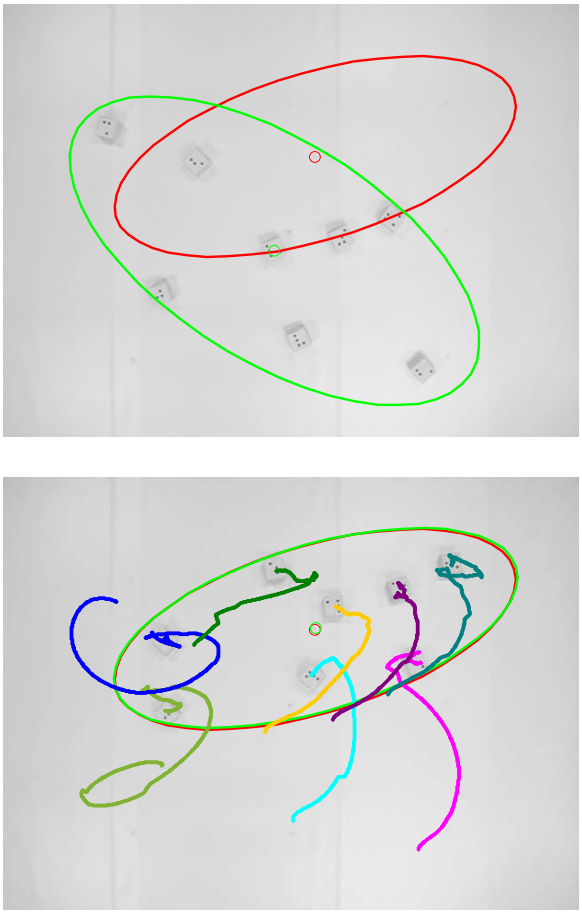

FIGURE 0.4.1. (Top) The initial and desired configurations of the swarm. (Bottom) A trace of the motion of the robots.

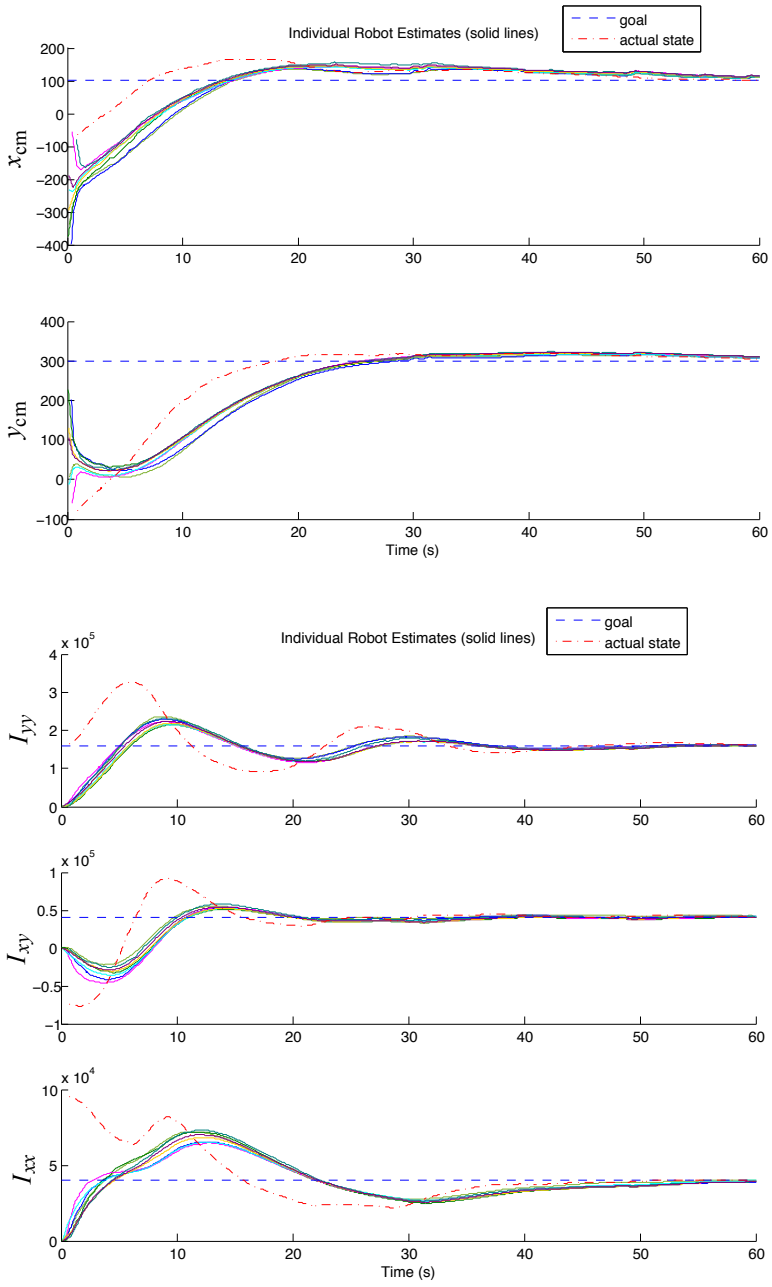

FIGURE 0.4.2. Time histories of the goal swarm moments, the actual swarm moments, and each robot's estimate of the swarm moments. (Top) First moments. (Bottom) Second moments.

0.4.2. Effect of Poorly Selected Gains. To explore the effect of controller gains, we increased the controller gains Γ by a factor of 10. Because robot speeds are usually saturated anyway, the primary effect of this uniform increase in gains is increased oscillation when the robots get close to the goal state. Figures 0.4.3 and 0.4.4 show the experimental behavior.

Increasing the estimator gains eventually causes the discrete-time estimators to become unstable, leading to unstable swarm motion.

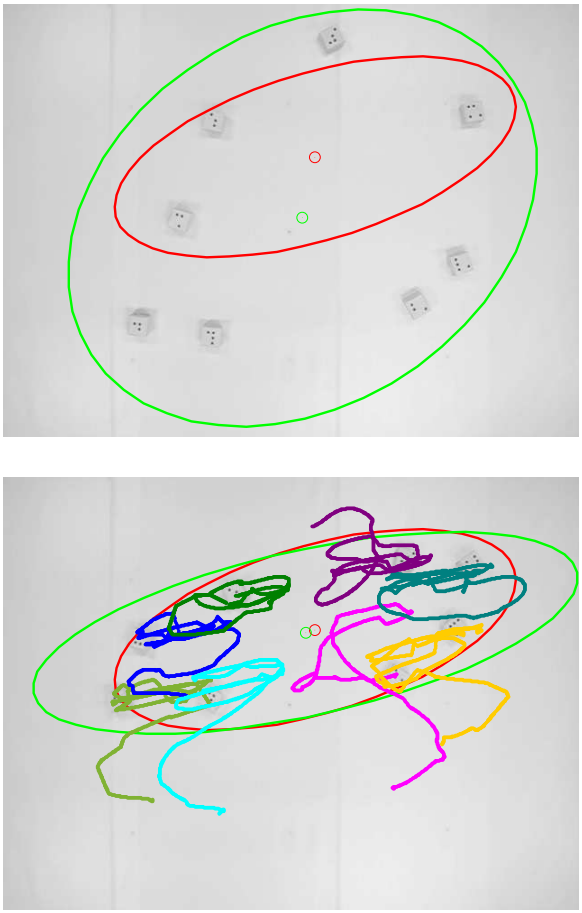

FIGURE 0.4.3. (Top) The initial and desired configurations of the swarm. (Bottom) A trace of the motion of the robots with large controller gains.

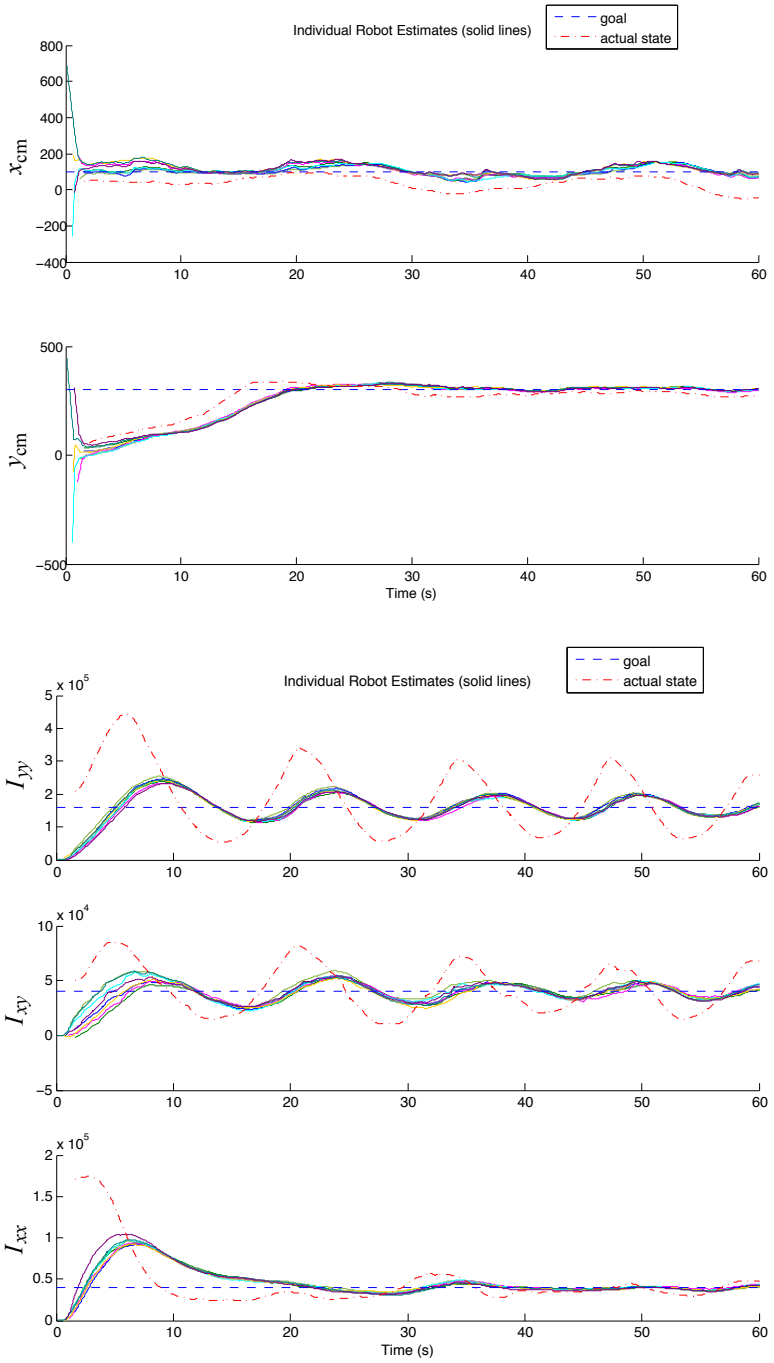

FIGURE 0.4.4. Time histories of the goal swarm moments, the actual swarm moments, and each robot's estimate of the swarm moments with large controller gains. (Top) First moments. (Bottom) Second moments.

0.4.3. Limited Communication Radius: 750 mm. This experiment uses the same conditions as the first, except that the robots are given a maximum communication radius of 750 mm. Every robot discards packets received from robots outside of this radius. The decreased connectivity and transients induced by the breaking and establishment of communication links leads to increased noise in the estimates and a slower settling time. The performance is shown in Figures 0.4.5 and 0.4.6.

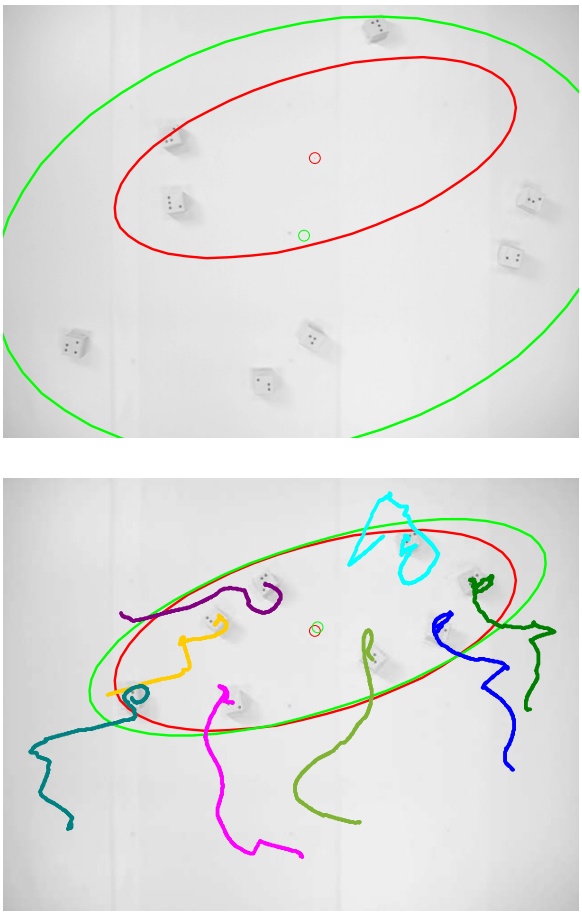

FIGURE 0.4.5. (Top) The initial and desired configurations of the swarm. (Bottom) A trace of the motion of the robots with communication radius set to 750 mm.

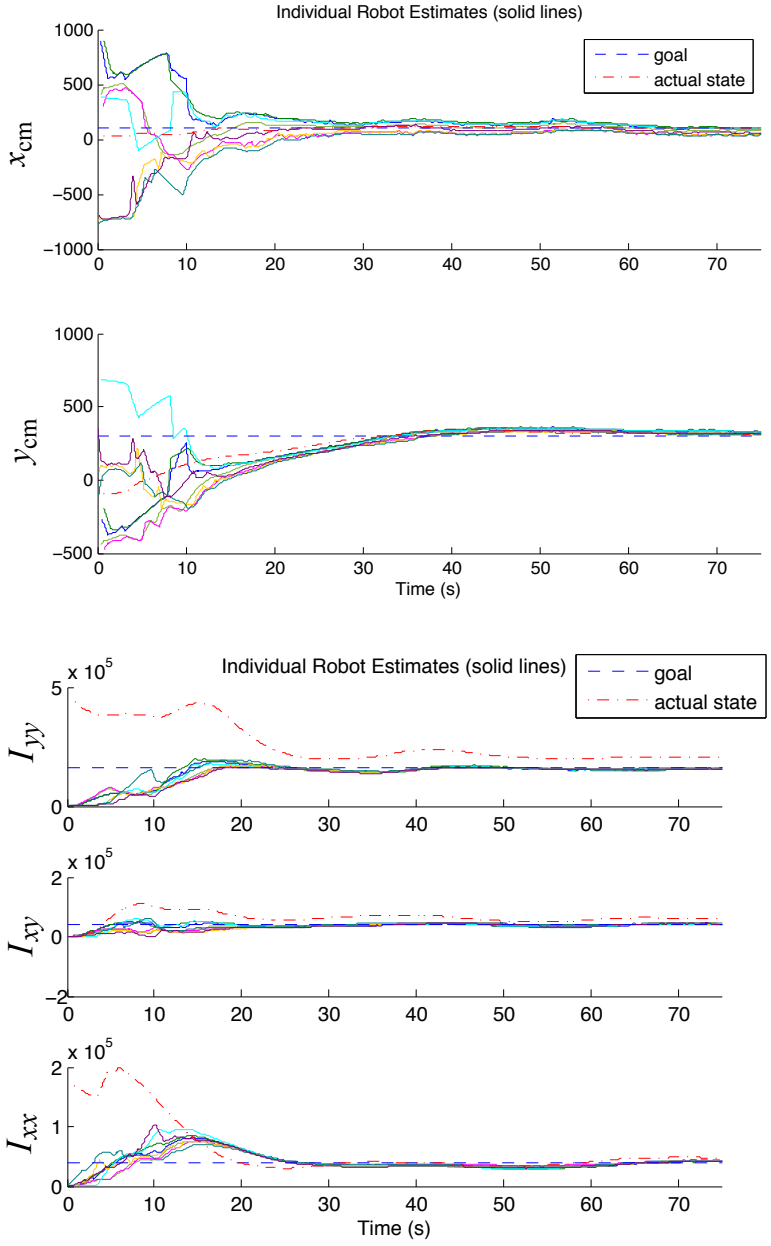

FIGURE 0.4.6. Time histories of the goal swarm moments, the actual swarm moments, and each robot's estimate of the swarm moments with communication radius set to 750 mm. (Top) First moments. (Bottom) Second moments.

0.4.4. Limited Communication Radius: 500 mm. When the communication radius is decreased to 500 mm, the performance of the swarm degrades further. The continual formation and disintegration of communication links creates transients that prevent the swarm from settling smoothly. Figures 0.4.7 and 0.4.8 show the results of this experiment.

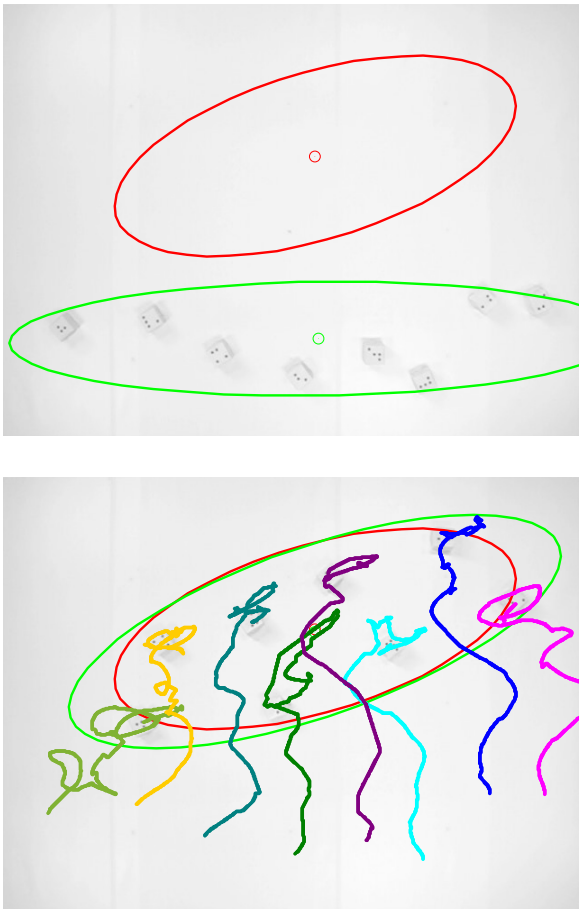

FIGURE 0.4.7. (Top) The initial and desired configurations of the swarm. (Bottom) A trace of the motion of the robots with communication radius set to 500 mm.

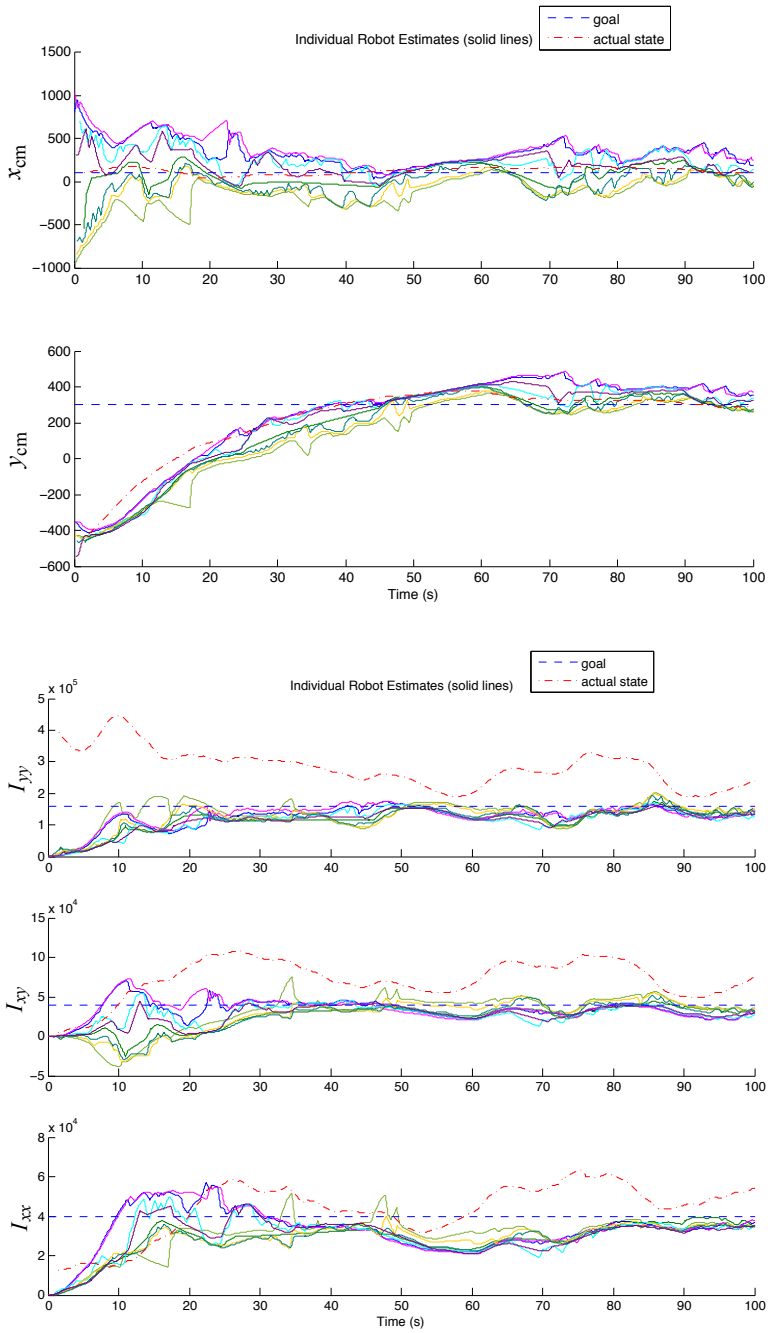

FIGURE 0.4.8. Time histories of the goal swarm moments, the actual swarm moments, and each robot's estimate of the swarm moments with communication radius set to 500 mm. (Top) First moments. (Bottom) Second moments.

0.4.5. Morphing Formation. In the next experiment, a sequence of goal states is sent to the swarm. The result is to move the swarm through a bottleneck, as marked by black virtual obstacles in Figure 0.4.9. We used the following sequence of formation moments:

(1) $f_1^\star = [-800 \text{ mm}, 0 \text{ mm}, 30000 \text{ mm}^2, 0 \text{ mm}^2, 150000 \text{ mm}^2]^T$
(2) $f_2^\star = [-600 \text{ mm}, 0 \text{ mm}, 50000 \text{ mm}^2, 0 \text{ mm}^2, 50000 \text{ mm}^2]^T$
(3) $f_3^\star = [0 \text{ mm}, 0 \text{ mm}, 160000 \text{ mm}^2, 0 \text{ mm}^2, 20000 \text{ mm}^2]^T$
(4) $f_4^\star = [600 \text{ mm}, 0 \text{ mm}, 50000 \text{ mm}^2, 0 \text{ mm}^2, 50000 \text{ mm}^2]^T$
(5) $f_5^\star = [800 \text{ mm}, 0 \text{ mm}, 30000 \text{ mm}^2, 0 \text{ mm}^2, 150000 \text{ mm}^2]^T$

Figure 0.4.9 shows the starting positions and the successive goals of the robots, as well as a trace of the paths of the robots. Figure 0.4.10 shows the actual and estimated moments as a function of time.

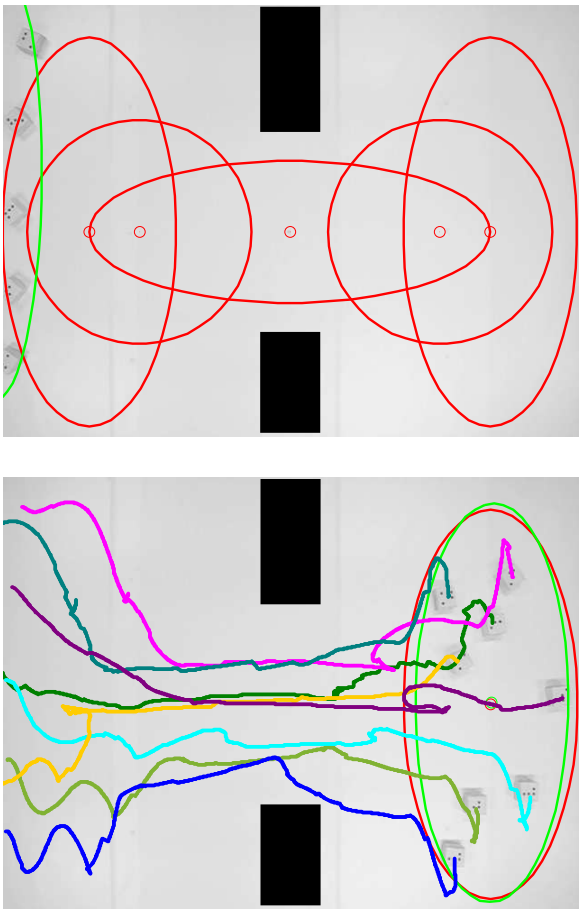

FIGURE 0.4.9. (Top) The initial and desired configurations of the swarm. (Bottom) A trace of the motion of the robots during target morphing.

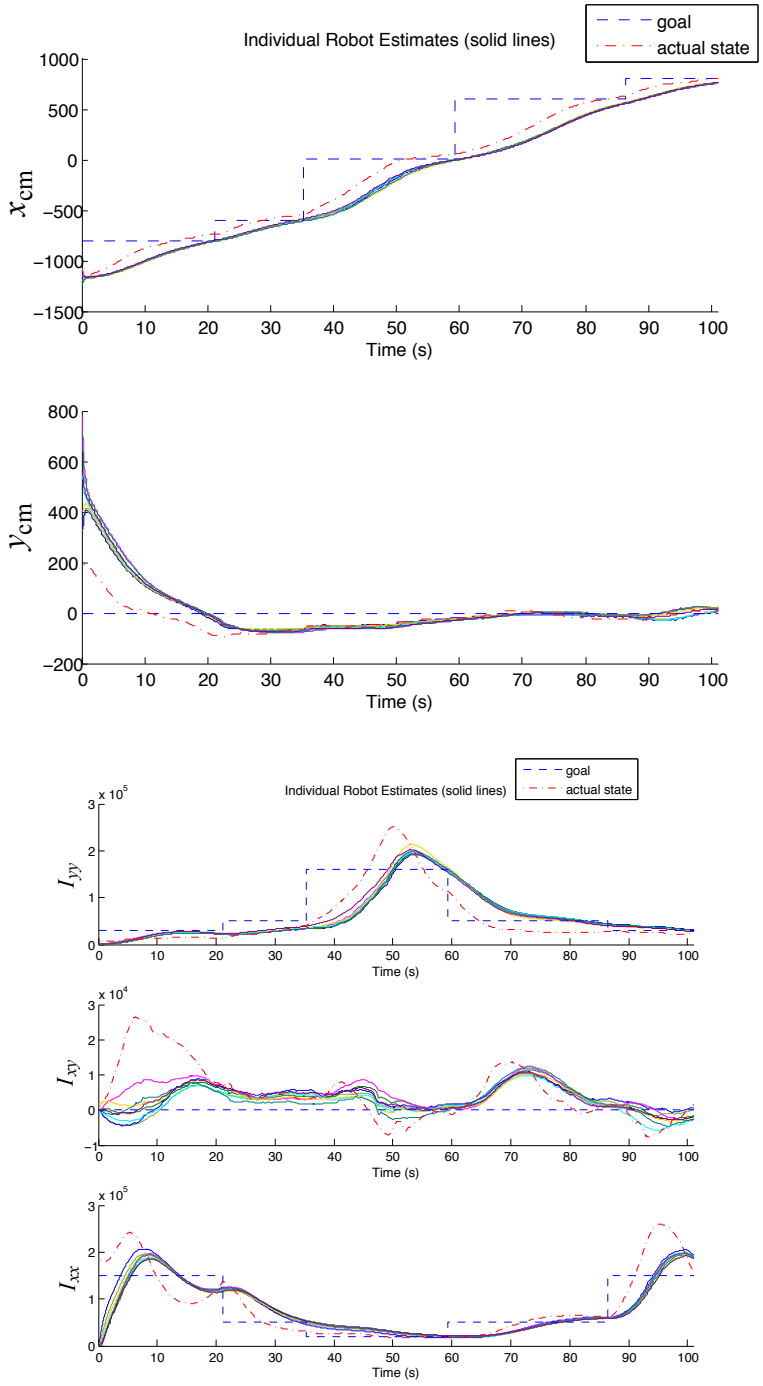

FIGURE 0.4.10. Time histories of the goal swarm moments, the actual swarm moments, and each robot's estimate of the swarm moments during target morphing. (Top) First moments. (Bottom) Second moments.

0.5. Conclusion and Future Work

Our experimental results demonstrate that a group of mobile robots can achieve a desired formation, as described by a set of moments, without the need for any central coordination. We achieved this result by adapting the theory presented in [**6, 16**] for implementation on real hardware. Despite this success, we had to make a number of choices to convert the theory to experiment, and many open questions remain.

In our implementation, only packets received in the past T_2 seconds are incorporated into the state estimate update. Intermittently dropped packets, or a communication link that is newly broken or established because of a communication radius limit, can therefore lead to high-frequency variations in a robot's state estimate. This is particularly noticeable in the 500 mm communication radius experiment in Figure 0.4.8. An alternative would be to keep estimates received during the past T seconds, $T > T_2$. Each time a new estimate is received from a particular robot, it replaces that robot's previous estimate. If a robot has not been heard from in the past T seconds, its estimate is deleted. Choosing $T > T_2$ would create a low-pass effect, reducing variations in each robot's estimate and making the estimates less sensitive to changes in network topology. This would come at the expense of responsiveness of the estimator—if T is chosen large, then stale information, or even information from a robot that has since been removed from the swarm, may be included in the estimator updates.

A related issue is the implementation of information-diffusion broadcasting. Our approach is extremely simple; future research could focus on more advanced network protocols that attempt to maximize network bandwidth and limit packet loss.

We chose gains for the consensus estimator empirically, based on experiments. While there exist guidelines for choosing stable estimator gains for continuous-time [**5, 7**] and synchronous discrete-time [**10**] estimators, such guidelines do not exist for asynchronous discrete-time estimators. Future work should address choices of gains that yield stability under worst-case conditions and good performance under typical conditions.

We are currently adapting our robots to allow them to sense properties of the environment. Each robot has an upward-pointing color sensor, and an overhead computer projector projects virtual dynamic environments onto the floor. Color from the projector simulates a time-varying function of position, such as temperature or chemical concentration. This new setup will allow us to perform experiments in control for optimal data collection and decentralized environmental modeling [**8**].

Acknowledgments. We thank the several Northwestern undergraduates who have led the continued development of the robot swarm experiment, including Ryan Cook, Sam Bobb, Jonathan Lee, and Neal Ehardt. This work was supported by the National Science Foundation and the Office of Naval Research.

Bibliography

[1] Y. I. Abdel-Aziz and H. M. Karara, *Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry.*, Proceedings of the Symposium on Close-Range Photogrammetry (1971), 1–18.

[2] He Bai, Randy A. Freeman, and Kevin M. Lynch, *Robust dynamic average consensus of time-varying inputs*, IEEE International Conference on Decision and Control, 2010.

[3] Calin Belta and Vijay Kumar, *Abstraction and control for groups of robots*, IEEE Transactions on Robotics **20** (2004), no. 5, 865–875.

[4] Digi International, *Xbee / xbee-pro rf modules*, v1.xex ed., 2009.

[5] Randy A. Freeman, Thomas R. Nelson, and Kevin M. Lynch, *A complete characterization of a class of robust linear average consensus protocols*, American Control Conference, 2010.

[6] Randy A. Freeman, Peng Yang, and Kevin M. Lynch, *Distributed estimation and control of swarm formation statistics*, American Control Conference, 2006.

[7] ———, *Stability and convergence properties of dynamic consensus estimators*, IEEE International Conference on Decision and Control, 2006, pp. 338–343.

[8] Kevin M. Lynch, Ira B. Schwartz, Peng Yang, and Randy A. Freeman, *Decentralized environmental modeling by mobile sensor networks*, IEEE Transactions on Robotics **24** (2008), no. 3, 710–724.

[9] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stéphane Magnenat, Jean-Christophe Zufferey, Dario Floreano, and Alcherio Martinoli, *The e-puck, a robot designed for education in engineering.*, Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, 2009, pp. 59–65.

[10] Thomas R. Nelson, *Decentralized worst-case estimation and control in mobile sensor networks*, Ph.D. thesis, Northwestern University, Evanston, Illinois, June 2010.

[11] Reza Olfati-Saber and Jeff S. Shamma, *Consensus filters for sensor networks and distributed sensor fusion*, IEEE International Conference on Decision and Control, December 2005, pp. 6698–6703.

[12] Demetri P. Spanos, Reza Olfati-Saber, and Richard M. Murray, *Dynamic consensus on mobile networks*, IFAC World Congress, 2005.

[13] Peng Yang, Randy A. Freeman, Geoffrey J. Gordon, Kevin M. Lynch, Siddhartha Srinivasa, and Rahul Sukthankar, *Decentralized estimation and control of graph connectivity in mobile sensor networks*, American Control Conference, 2008.

[14] Peng Yang, Randy A. Freeman, Geoffrey J. Gordon, Kevin M. Lynch, Siddhartha S. Srinivasa, and Rahul Sukthankar, *Decentralized estimation and control of graph connectivity for mobile sensor networks*, Automatica **46** (2010), no. 2, 390–396.

[15] Peng Yang, Randy A. Freeman, and Kevin M. Lynch, *Distributed cooperative active sensing using consensus filters*, IEEE International Conference on Robotics and Automation, 2007.

[16] ———, *Multi-agent coordination by decentralized estimation and control*, IEEE Transactions on Automatic Control **53** (2008), no. 11, 2480–2496.

[17] Minghui Zhu and Sonia Martínez, *Discrete-time dynamic average consensus*, Automatica **46** (2010), no. 2, 322–329.