**Swarm Demonstration Hardware System: Applied to Maintaining a Wireless Sensor Network**

Christiana Chamon1, Rachel Dunn1,2, Maria Ciara Lalata1, and Mable Wan1

Mentor: Dr. Aaron Becker1

1 Department of Electrical and Computer Engineering

2 Department of Biomedical Engineering

University of Houston

Houston, TX 77204-4005

**Introduction**

Swarm robotics is a field of robotics wherein multiple mini robots work together to complete specific tasks. This project seeks to maintain the batteries of a swarm of sensor nodes using a few charging robots. Each of our robots is relatively cheap, small, and easy to use and and assemble, making them attractive prospects for outreach or educational purposes. This document contains the instructions necessary to recreate our project. Our files can be found at <https://github.com/mable-won/SeniorDesign>, and our video demonstrations can be found at <https://youtu.be/Rzd7-cDX41M>. Our project was inspired by the open-source project “microMVP: A Portable, 3D-­Printing Enabled Multi­-Vehicle” available at <http://arc.cs.rutgers.edu/mvp/index.html>. If you have any questions after looking over this document or the code, feel free to contact Rachel Dunn at redunn416@gmail.com.

**Hardware**

*Parts*

We created 12 total robots, 4 of which were charging robots, and 8 of which were sensor nodes. Every vehicle, regardless of type, requires the items listed in Table 1. Communicating with the robots will require 2 extra XBee Series 1 modules. Charging robots will require the parts listed in Table 2, and sensor nodes will require the parts listed in Table 3. Note that one package of coils will create one charging robot and one sensor node. Additional components necessary for our project are listed in Table 4. Prices and links were posted April 2017 and may have changed since then. Prices assume bulk purchasing for building 10-25 vehicles. Therefore, for a total of 12 robots, consisting of 4 charging robots and 8 sensor nodes, in addition to the supplementary components, the total cost of our project is $1,517.49.

Table 1. Vehicle Components

|  |  |  |
| --- | --- | --- |
| Part and link | Description | Cost (USD) |
| [32x7mm wheels](https://www.pololu.com/product/1088) | A pair of Pololu wheels that can be snapped onto 3mm D-shaft gearmotors. | 3.95 |
| [Battery](https://www.sparkfun.com/products/13851) | 400mAh Li-ion battery. | 4.95 |
| [Caster ball](https://www.sparkfun.com/products/8909) | Metal ball bearing supporting the front of the robot shell. | 2.95 |
| [Fio v3 board](https://www.sparkfun.com/products/11520) | Sparkfun fio v3 with Arduino support, built-in XBee interface, and battery charging support. | 33.20 |
| [Gearmotors (x2)](https://www.pololu.com/product/1511) | Pololu 120:1 mini plastic gearmotor with 90­°, 3mm D-Shaft output. | 2x 4.95 |
| [Motor Driver Carrier](https://www.pololu.com/product/2135) | Pololu DRV8835 dual motor driver carrier board. | 3.89 |
| [Vehicle shell](https://github.com/mable-won/SeniorDesign) | 3D-printed custom shell designs for charging robots and sensor nodes. | ~0.80 |
| [XBee series 1](https://www.sparkfun.com/products/11215) | XBee series 1 (802.15.4) with trace antenna. | 24.95 |
| **Total** |  | 84.59 |

Table 2. Charging Robot Additional Components

|  |  |  |
| --- | --- | --- |
| Part and Link | Description | Cost (USD) |
| [Gripper](https://www.sparkfun.com/products/13176) | Micro gripper kit A with a straight mount. | 5.95 |
| [Servo](https://www.amazon.com/TowerPro-SG90-Mini-Servo-Accessories/dp/B001CFUBN8) | TowerPro SG90 9G mini servo. | 3.79 |
| [Wireless power transfer module](http://www.sunrom.com/p/wireless-power-transfer-modules) | Transmitter coil of the wireless power transfer module. | ½x 22.92 |
| **Total** |  | 21.20 |

Table 3. Sensor Node Additional Components

|  |  |  |
| --- | --- | --- |
| Part and Link | Description | Cost (USD) |
| [Capacitor](http://www.mouser.com/ProductDetail/Eaton/PM-5R0V105-R/?qs=yoyprvkDL1kslTgYewWZLA%3D%3D&gclid=Cj0KEQjw8tbHBRC6rLS024qYjtEBEiQA7wIDeWXap7TGcTSWDk-IN-Ulq3TPNG7mtutNh3L4vXxDkC4aApFJ8P8HAQ) | 1 [F] 5 [V] super capacitor. | 5.74 |
| LEDs | 1 [red LED](http://www.mcmelectronics.com/product/25-5662?green=356C6B55-9BB1-5594-9755-DC78D29D29F7) and 1 [green LED](http://www.mcmelectronics.com/product/25-5648?green=356C6B55-9BB1-5594-9755-DC78D29D29F7) (optional). | 0.38 |
| Resistors | 1x 1 [MΩ] and 2x 100 [Ω] (variation acceptable). | ~0.20 |
| [Wireless power transfer module](http://www.sunrom.com/p/wireless-power-transfer-modules) | Receiver coil of the wireless power transfer module. | ½x 22.92 |
| **Total** |  | 17.78 |

Table 4. Supplementary Components

|  |  |  |
| --- | --- | --- |
| Part and Link | Description | Cost (USD) |
| [Microphone stand](https://www.amazon.com/dp/B019NY2PKG) | Basic microphone stand for mounting the USB video camera (optional) | 15.95 |
| [Super glue](https://www.amazon.com/Gorilla-Super-Glue-Gel-15/dp/B00CJ5EO2E) | Gorilla super glue, used for assembling the robot shell (optional). | 4.58 |
| [USB extension cord](https://www.amazon.com/AmazonBasics-Extension-Cable--Male--Female/dp/B00NH11PEY) | Extension cable for connecting USB video camera to computer (optional). | 5.79 |
| [USB micro cord](https://www.amazon.com/dp/B01EK87T9M/?ref=sxts_snpl_3_0_2958654442&qid=1492489112&pf_rd_m=ATVPDKIKX0DER&pf_rd_p=2958654442&pf_rd_r=GRHW6NJ71Z9ZK6V28NT4&pd_rd_wg=2KALY&pf_rd_s=desktop-signpost&pf_rd_t=301&pd_rd_w=vhv8Z&pf_rd_i=micro+usb+cable&pd_rd_r=8BNXR3M15WX273509F50) | Cable for uploading code to the fio (optional). | 4.41 |
| [USB webcam](https://www.amazon.com/Ipevo-Ziggi-HD-High-Definition-Document-CDVU-06IP/dp/B01530XGMA) | Ipevo high-definition USB document camera. | 99.00 |
| [XBee explorer dongle (x2)](https://www.sparkfun.com/products/11697) | XBee explorer dongle, used for configuring XBees and wireless communication from a computer. | 2x 24.95 |
| **Total** |  | 179.63 |

*Assembly*

The shells for the charging robots and sensor nodes are composed of four files each, available on our Github website. We used an Ultimaker 2+ 3D printer to print them out, but any 3D printer should be able to print the robot shells. The nozzle we used was a 0.4 [mm], and the material used was PLA--and in a variety of colors, but we discovered darker colors worked better for the image processing. The software used to design these shells was Autodesk Fusion 360, or the Mac version of Autodesk Inventor. Our .f3d files and our .stl files are available on our Github website. The .stl files can then uploaded into a program called Cura, which then extracts the .stl files into .gcode files ready to be 3D printed. Once printed, the four large pieces must be glued together, and the smaller overhang pieces should be glued to the front of the shell to hold the coil in place. The caster ball should be glued on to the bottom of the assembly as well (using only the ball assembly and the thicker 1/8" spacer). Figure 1 shows the assembled charging robot shell, and Figure 2 shows the assembled sensor node shell. We used sticky tack to keep the wheels and gripper in place.

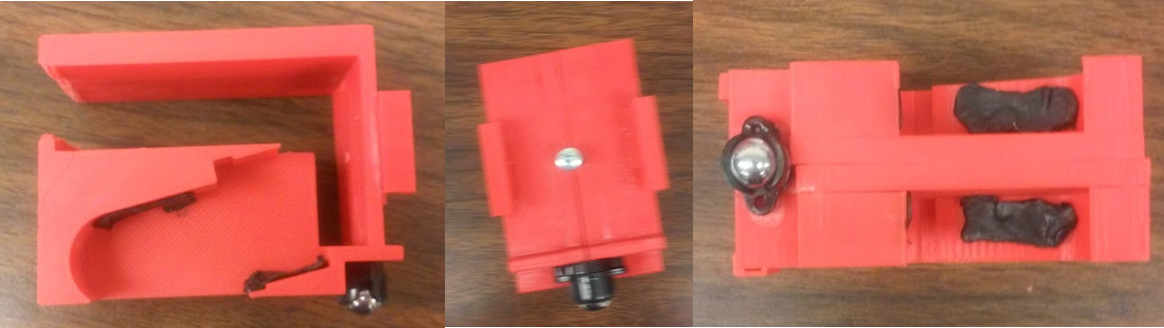


Figure 1. Charging robot assembled shell.

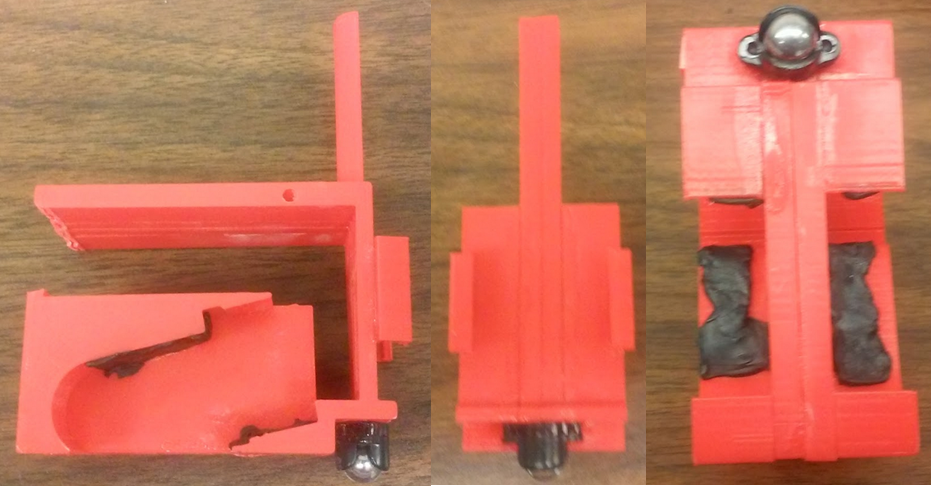


Figure 2. Sensor node assembled shell.

For every vehicle, regardless of type, the gearmotors and dual motor driver carriers must be connected as shown in Figure 3 below. Take note that for charging robots, the positive and negative leads of the transmitter coil (the smaller one) must also attach to the pins near the battery (where VIN and GND of the motor driver carrier attach). The connections to the gearmotors are very weak and may break off unexpectedly, so we recommend securing the connections with a glue gun. Also, placing electric tape between the Fio and the motor driver carrier prevents shorts from occurring.

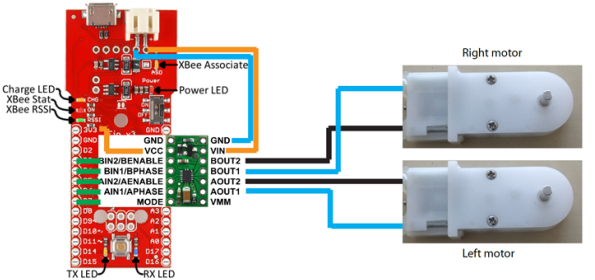


Figure 3. Motor connections for all robots.

For sensor nodes, the extra connections should be as shown in Figure 4a. The 1 [F] capacitor, 1 [MΩ] resistor, and the leads from the larger chip of the coils should be connected in parallel across A2 and A0. Then ground A0. The 100 [Ω] resistors, soldered in series with the longer leg of the LEDs, should be soldered to the D9 and D11 pins for the red and green LEDs, respectively. The shorter leg of the LEDs should be grounded.

For the charging robots, the extra connections should be as shown in Figure 4b. The brown wire of the servo should be connected to ground, the red wire should be connected to a 3V3 pin, and the orange wire should be connected to D10. The coil with the smaller chip should be soldered to the 5[V] input and ground next to the battery (where VIN and GND from the dual motor driver carrier are connected).

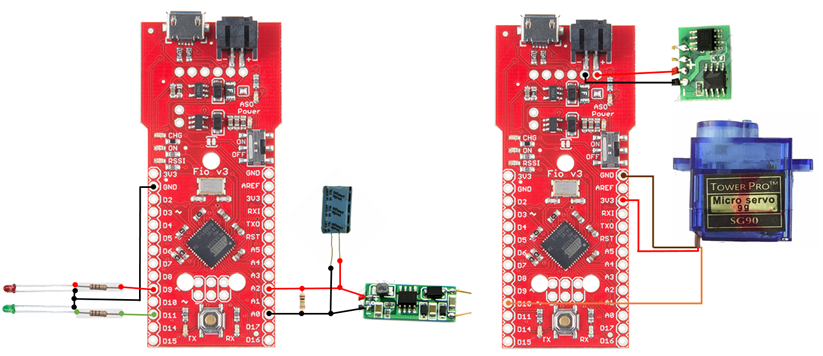


Figure 4. Sensor node extra connections (left) and charging robot extra connections (right).

After soldering the main components, the wheels should be pressed onto the motors, and the configured XBees attached to the corresponding Fio. Once the coil is in place, a screw can be drilled into the shell for faster power transfer. The Fio can operate on battery power or through the microUSB cord. Plugging both into the Fio will charge the battery. A word of caution: it is known that the Fio v3 board has a current leak even when turned off. Therefore, the battery should be disconnected after each use of the vehicle. Figure 5 shows the fully soldered robots. From there, it should be relatively simple to put the robotic components into the shells. Figures 6 and 7 show our fully assembled charging robot and sensor node robot, respectively.



Figure 5. Fully soldered charging robot (left) and sensor node (right).

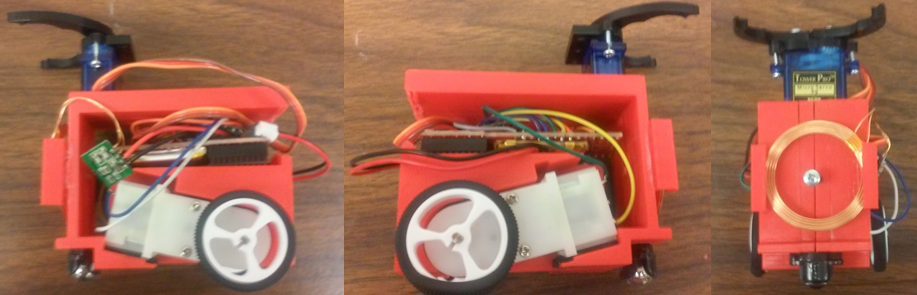


Figure 6. Left, right, and front views of the fully assembled charging robot.

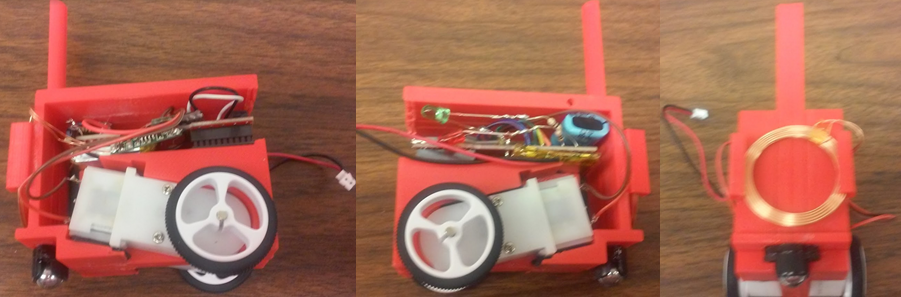


Figure 7. Left, right, and front views of the fully assembled sensor node.

*Camera Platform and Arena*

The webcam can be mounted on a microphone stand or some similar fixture using masking tape. The camera should be around five feet away from the ground, facing down, and connected to the computer running the charging robot code. Each vehicle should have a ladybug-shaped identification tag containing the same number of dots as the carID. Our custom tags cover all numbers through 12 and are available for printing at our Github website. The environment should have non-carpeted, even flooring, covered with non-reflective black butcher paper, which can be taped onto the floor.

**Software**

*Installation*

The following applications and packages must be installed for any computer running our project. All programs are free downloads, except for Matlab which requires a license to use. Installation instructions are available on the developer’s website.

[Arduino IDE](https://www.arduino.cc/en/main/software)

[Arduino: Sparkfun board](https://github.com/sparkfun/Arduino_Boards)

[Arduino: XBee library](https://github.com/andrewrapp/xbee-arduino)

[Matlab](https://www.mathworks.com/downloads/)

[MATLAB Support Package for USB Webcams](https://www.mathworks.com/matlabcentral/fileexchange/45182-matlab-support-package-for-usb-webcams)

[XCTU](https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu)

*XBee Configuration*

Use XCTU to configure the XBees and test communications. Before configuration, update all XBees to the newest firmware: XB24, XBee 802.15.4, 10ef (or whichever is newest). All XBees should communicate on the same channel and PAN ID (e.g. 12 and 2048, respectively). All robots and coordinators should have a distinct MY address. Our code requires coordinator 1’s address to be 1874, and coordinator 2’s address to be 000D. If you want to change them, you must also change the Arduino files so that the robots send packets to the right coordinator. The destination address for sensor nodes should be coordinator 1’s MY address, and the destination address for the charging robots should be coordinator 2’s MY address. The coordinators’ destination address should be 0. You can use NI to input a string to identify that XBee. We numbered our robots SN1-8 and CR9-12. The baud rate should be set to 57600, and the AP enable should be set to API enabled with PPP. All other fields should remain at their default values. After configuration, the XBees are ready to communicate.

*Arduino*

Under Tools, set the board to Sparkfun Fio v3 and the port to the one that connects to the Fio using the microUSB cord. If the port is unavailable, follow the instructions [here](https://learn.sparkfun.com/tutorials/pro-micro--fio-v3-hookup-guide). Upload the chargingRobot-v1 to every charging robot and the sensorNode-v1 to every sensor node, changing the carID for each vehicle. Sensor nodes should be numbered 1-8, and charging robots should be numbered 9-12. For charging robots, test each gripper first uploading the Sweep program under Servo examples to your Fio to find good values for opened and closed (values should have a difference of about 60). Change the values of the openServo and closeServo functions to the ones tested with Sweep. Movement and gripper functions can be tested with test.m in Matlab. If a robot moves backward instead of forward, in the Arduino file, go to the processCommand function, and switch sign of the last two numbers in the assignment of leftThrust and rightThrust (i.e. 1 becomes -1 and -1 becomes 1).

*Camera Tracking*

The camera tracking program uses the IDs in the attached file “Chevrons.ppt.” The tracking program can only track the ladybugs when the background is entirely another color than white. The objects that have the ladybug identification tags also cannot be reflective or else the webcam detects the reflection as another identification tag.

The tracking program is robust because it can detect the ladybugs on any background, given that it is not white or a reflective color. To set up the tracking program on another background or from another height, the user needs to adjust the configurations on the webcam. The easiest method is to edit the webcam configurations from the matlab command window. The main configuration to adjust is **focus.** The example below shows how to set up the webcam. These directions are directly written to the matlab terminal.

**vid=webcam(#);** %insert 1 or 2

**vid** % Shows current webcam configurations.

**preview(vid);**  % Real time view from the webcam.

**vid.Resolution**= ‘LxW’; %Change resolution based on height. We used ‘1024x768’.

**vid.Focus= N;** % N is adjustable from 0-190. The further away the webcam is, the lower the focus.

**vid.Contrast=N;** % This clears up vague lines.

Running the program TrackingChevron\_noFunction.m performs a real time image processing of camera images. Figure 8 shows how the program tracks the ladybugs. The blue asterisk is the centroid. The blue number is the identification of that ladybug and is also the number of dots on the tag. The red asterisk indicates the direction of the ladybug. This program is a function and the output is an array called **outVector**. This vector is a 2-D array and it contains, for each robot, the ladybug ID (number of dots), x-centroid, y-centroid, orientation (0-360°). The array is not organized by ladybug ID. Below is an example:

**outVector** = [5, 35, 54, 90, 3, 56, 102, 178 ];

**outVector** states that ladybug-5 is at location (35, 54) in the 90° orientation. Ladybug-3 is at location (56, 102) in the 178° orientation.

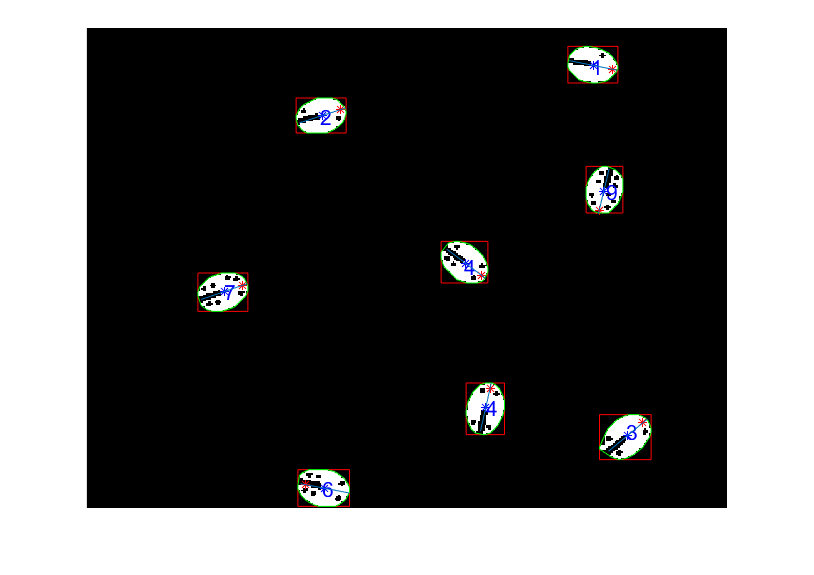


Figure 8. Image processing of ladybug identification tags.

*Matlab*

To test that your robots have been correctly assembled, open the file test.m. Uncomment the test you wish to run, and change the serial port number to the one the to which XBee explorer dongle is connected (this number can be found in XCTU or through Control Panel=>Devices and Printers for Windows operating systems).

To run the sensor node code, which coordinates the movement of all sensor nodes by using a random walk model with drift, open sensorNodev2, changing your com port to the correct number and any of the tunable parameters you wish. Then run SNGUI.m. The toggle button allows you to start and stop your program whenever you wish, but remember to stop the simulation before exiting the GUI (see note in sensorNodev2.m). The simulation, shown in Figure 9, will be shown in real time as commands are sent to your robots. Note that as long as the dongle is connected, the robots do not need to be operational to view the simulation. To align the robots with the simulation, the initial positions must be the same. Therefore, evenly space out the sensor nodes on the left side of the camera or manually change the starting positions in SensorNodev2. In the current iteration of the code, the sensor nodes follow the random walk with drift model. However, the collision avoidance still needs work.



Figure 9. Sensor node movement simulation.

To run the charging robot code, make sure that CRTracking and CRMain are in the same directory. First, make sure that the camera sees all the sensor nodes and charging robots. This can be done by running TrackingChevron\_noFunction.m. Then, run CRMain and the program should be able to have the camera correctly identify the carID of each of the sensor node and charging robot (an image processing window similar to Figure 8 should pop up), assign which charging robot will go to each sensor node, have the charging robots move towards their assigned sensor node, and have the charging robot charge the sensor node. The sensor nodes all have an LED system composed of a green and a red LED that indicates the charging status. For a voltage less than 1.25 [V] the red LED remains on. For a voltage of 1.25 - 2.5 [V], the red LED blinks. For a voltage between 2.5 and 3.3 [V], the green LED blinks. And for a voltage of 3.3 - 5.0 [V], the green LED remains on.

The charging robot algorithm is a distance-based curvature algorithm that takes location and orientation information for both the current position of the charging robot and the sensor node from the camera input and outputs values to the left and right wheels. The algorithm also takes into account a moving target sensor node and the charging robot does follow the sensor node even when it moves. The algorithm, however, does not take into account an alignment algorithm where the charging robot will orient itself so that its head will align with the head of the sensor node. Figure 10 shows the simulation of the charging robot algorithm. To make this simulation, simply run MultipleChargingRobot. To show sensor node movement and that the charging robot does track a moving sensor node, the crosses indicate the locations of the charging robot with the black ones representing the initial location of the sensor node while the magenta ones represent the final location of the sensor node. The simulation is tested without the camera. To update the location without camera input, use the algorithm shown in UpdateLocation.

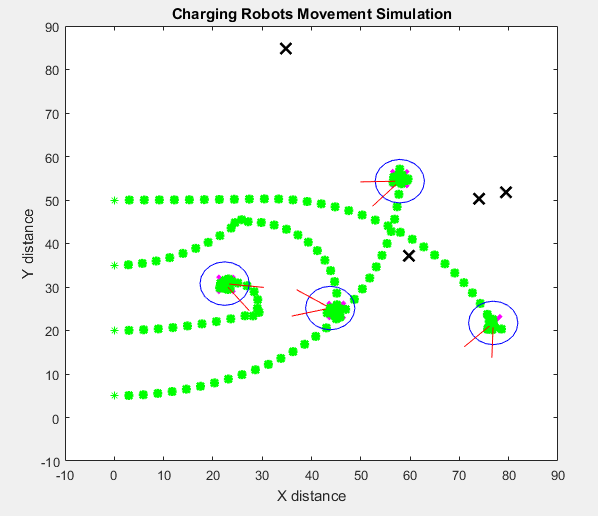


Figure 10. Charging robot movement simulation.

*Altering the Code*

To change the number of vehicles, the Arduino code for all vehicles must be updated, and the Matlab code for sensor nodes and charging robots must be updated. For the Arduino code, change the CRNumber, the number of charging robots, and SNNumber, the number of sensor nodes. In Matlab, for charging robots, change the values of the global variables CRNumber and SNNumber in CRMain. Also in CRMain, change the number of timers to the number of charging robots, using the correct carID that correlates to the timer. Also change the cases of the switch. For sensor nodes, change the value of SNNumber in sensorNodev2. In the provided code, vehicle thrusts are sent in 10 or 18 byte packages. The package is identified by the first and last byte. The middle 8 or 16 bytes are wheel thrusts for 4 or 8 vehicles. Making this package longer will then allow more vehicles to be added. Changing the first and last byte will change the way the Fios interpret the data. For example, an XBee broadcasting to sensor nodes will have a first and last byte of ‘C’ and ‘M’. Sensor nodes interpret this packet as wheel thrust commands. Charging robots do not recognize this byte combination. For their movement, the first and last byte will be ‘M’ and ‘C’, which sensor nodes do not recognize. In this way, one can send commands to specific vehicles for different functionalities. The slight downside with this approach is that as more and more vehicles are added, there can be interference between the XBees. This can be mitigated by limiting the number of XBees on the same channel.

**Acknowledgements**

We would like to thank Dr. Becker for sponsoring our project and for being our advisor. We are appreciative of Dr. Jingjin Yu for the base robot design and for being open to our questions. We would also like to thank Dr. Le, Dr. Lim, Dr. Trombetta, and Dr. Wosik for facilitating the senior design courses and for keeping us on track.