Project Title: Autonomous Self-Reconfigurable Modular Robots

Team Members: Caleb Gordon (CmpE)
Nicholas Heath (CmpE)
Nathan Seelye (EE)

Faculty Advisor: Yanfei Liu, Ph.D.

Date: December 9, 2016
# Table of Contents

Acknowledgements........................................................................................................... 5
Abstract................................................................................................................................. 6

## Section I: Problem Statement ......................................................................................... 7
  1.1 Requirements and Specifications: ................................................................................ 8
  1.2 Given Parameters or Quantities: .................................................................................. 9
  1.3 Design Variables: ........................................................................................................... 9
    1.3.1 Hardware: .................................................................................................................. 9
    1.3.2 Software: ..................................................................................................................... 9
    1.3.3 System Conditions: .................................................................................................... 9
  1.4 Limitations and Constraints: ........................................................................................ 10
  1.5 Additional Considerations: .......................................................................................... 10

## Section II: Detailed Design of the Selected Conceptual Design ....................................... 11
  2.1 Microcontroller Configuration ...................................................................................... 12
    2.1.1 Introduction to Bluno Nano ....................................................................................... 12
    2.1.2 Communication .......................................................................................................... 12
    2.1.3 Bluno Nano Connections .......................................................................................... 13
  2.2 Sensors .......................................................................................................................... 14
    2.2.1 Image sensing ............................................................................................................ 14
    2.2.2 Ultrasonic sensing ...................................................................................................... 15
  2.3 Motors, Battery and Wheels .......................................................................................... 16
    2.3.1 Motor Space Calculations .......................................................................................... 16
    2.3.2 Motor and Wheel ........................................................................................................ 16
    2.3.3 Battery ....................................................................................................................... 17
    2.3.4 Power ......................................................................................................................... 18
    2.3.5 Magnets ..................................................................................................................... 18
  2.4 Physical Build ................................................................................................................ 19

## Section III: Revisions of Final Implemented Design ....................................................... 23
  3.1 Motors and Motor Controller ....................................................................................... 24
    3.1.1 Motors ....................................................................................................................... 24
    3.1.2 Motor Controllers ...................................................................................................... 24
  3.2 Magnets and Orientation .............................................................................................. 25
“testing_BLE_command.ino” ................................................................. 54
Source Code to Test Ultrasonic Sensors .............................................. 57
“Ultrasonic_test.ino” ........................................................................ 57
Acknowledgements

We would like to thank our primary advisor Dr. Yanfei Liu for allowing us to participate in this project. Throughout the course of this project, she has been a very valuable resource for guidance and feedback. Without her help, this project would not have been possible. We would also like to thank our co-advisors Dr. Carlos Pomalaza-Ráez, Dr. John Licato and Dr. Zhuming Bi for providing their support in this project as well.

We would also like to thank Bill Westrick for volunteering to help us with this project. He has provided us with some ideas on specific parts of the project and even brought some components in at some of the meetings to observe and allowed us to have a more solid idea on the details of the project.

We would also like to thank Mr. Stephen Heath for providing his support in this project. He has over 40 years of mechanical experience and has helped us with some physical aspects of the robot such as installing the motors into the wheels.

Finally, we would also like to thank the IPFW Department of Electrical and Computer Engineering for sponsoring this project.
Abstract

One of the most popular subjects in computer and electrical research is artificial intelligence. Artificial intelligence is a field of study which studies how computer systems can be designed to exhibit intelligent behavior such as that of a human. Researchers most commonly study this subject through the use of robots. Robots exhibit intelligent behavior through the use of sensors and programming within its central control unit.

Modularity is one of the more important topics in robotics, being that artificial intelligence is commonly seen in robots that are able to move around and perform physical tasks. The main goal of this project is to create a self-reconfigurable modular robot that can assemble or disassemble with minimal human interaction. The purpose of creating this robot is to help researchers study how artificial intelligence works. The robot should be able to assemble or disassemble based off of user-defined programming, have an array of sensors, operate via a wireless protocol, move across a surface with minimal restriction, and be powered by a rechargeable power supply.
Section I: Problem Statement
The goal of this project is to design a modular robotic platform that can automatically assemble and reconfigure itself on command for researchers to further experiment with AI and reasoning. The platform should be equipped with an array of sensors, such as orientation sensing and module position relative to other modules to aid self-assembling and reconfiguration that is guided by high-level commands. The communication interface must also be wireless, using a standard protocol (i.e. Bluetooth, WiFi).

1.1 Requirements and Specifications:

The following requirements and specifications must be met for a successful project:

- **Central Control Unit** - The central controlling unit of the system must be programmable through a wireless system. The program data must be stored in nonvolatile memory. The central control unit must be able to either process sensory data or efficiently relay that sensor data for processing and then be able to communicate decisions to the other modules.

- **Power** - The system must contain some type of power source that will not restrict movement. The power source should be rechargeable and sufficient for a number of sensors. The power should be able to be switched on and off. Max charge should last 30 minutes.

- **Sensors** – The system should be able to collect data from various sensors to aid self-assembling and reconfiguration.

- **Modularity** - The modules should be able to disconnect and reconnect themselves based off of physical mechanisms on the modules.

- **Terrain** - The system must be able to move well on different surfaces. It must work on a hard floor, concrete, and carpeted surface.

- **Self-Assembly** - The modules should be able to connect from a disconnected state independently of manual aid. Modules should be able to assemble themselves from up to 2 feet between each module.

- **Self-Reconfiguration** - The modules should be able to form at least three different configurations with at least two modules in each configuration.

- **Communication** - The control unit of the modular robot system must be able to wirelessly transmit and receive data to a remote user interface. Modules can communicate to each other up to two feet apart. Remote control up to 3 feet.
1.2 Given Parameters or Quantities:

The following parameters are given in the design of the modular robotics system:

- **Budget** - The total cost of the project must be less than or equal to $1000.

- **Mobility** - The modules should be able to move in any direction on a flat plane and rotate for module orientation. The modules should be able to turn in left a left circle and a right circle.

- **Size** - The size of the module must be within the manufacturing capabilities of available 3D printing and CNC machining.

1.3 Design Variables:

1.3.1 Hardware:

- **Wireless Communication** - The robot will communicate with a controller via a wireless communication protocol.

- **Central Control Unit** - The central controller unit of the system will operate using an embedded system platform that is readily available for purchase.

- **Module Construction Materials** - Housings and other mechanical parts should be manufactured through common and precise processes such as 3D printing.

1.3.2 Software:

- **Software Compatibility** - The robot should be compatible with multiple programming languages that are commonly used among software engineers.

1.3.3 System Conditions:

- **Module Timing** - The modules that control the robot should be in sync to function properly, especially for movement and communication between other modules.
1.4 Limitations and Constraints:

- Durability - The robot should be able to move along various floor surfaces while resisting wear from movement across surfaces. The connections between modules should be durable to withstand strain from moving. The modules should be able to withstand damage from the max force another module can apply.

- Efficiency - Because the robot will be controlled wirelessly, it should be able to perform its intended tasks with minimal power consumption. The platform should be 70% efficient.

- Safety - Robot should be safe to handle (i.e. no sharp corners, shock hazards), regardless of operations performed.

1.5 Additional Considerations:

- Other types of modules - Using modules such as audio speakers or LEDs can add more programmable features to the robot.

- Battery Level Indicator - An indicator for the battery level would be useful to inform the user of when recharging is necessary.

- Smartphone Control - Implementation of an application downloadable by a smartphone to control the modular robots.
Section II: Detailed Design of the Selected Conceptual Design
Having rated the conceptual designs and some of the components, the project moved into the next stage in which the conceptual design is ready to be detailed.

2.1 Microcontroller Configuration

2.1.1 Introduction to Bluno Nano
The Arduino model that will be used for each module is the Bluno Nano. It is equivalent to the Arduino Nano but with a Bluetooth chip embedded on the board and a lack of ICSP (In-Circuit Serial Programming). This allows the module to communicate via Bluetooth without exhausting the only TX/RX pin available which could be used for other components that require serial communication. The Bluno Nano’s BLE transceiver has two different modes: AT and NORM. AT mode is used for configuring the Bluno Nano with a variety of settings while NORM mode is used for normal serial communication. It also has a transmission range of more than 20m.

2.1.2 Communication
The Bluno Nano in each module may communicate through either an iOS 7+ device, Android 4.3+ device with a BLE adapter, or a PC via Bluetooth Low Energy (BLE). In order for it to communicate with a PC, the PC must have a USB BLE link. Standard USB Bluetooth adapters are not compatible. There is some demo Android and iOS source code readily available for software programmers to get started with the Bluno Nano.
2.1.3 Bluno Nano Connections
Each pin on the Bluno Nano will be occupied with each corresponding device/connection shown below in Figure 13.

**Figure 3:** Bluno Nano Pin Connections (drawn in Xilinx)
Table 1 below shows a brief description of each pin that will be used on the Bluno Nano. The following pins are on the Bluno Nano but will not be used for the module: TX, RX, RST, A7-A1, and V3.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Type of Pin</th>
<th>Used For</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIN</td>
<td>Voltage Input</td>
<td>Power Supply</td>
</tr>
<tr>
<td>GND</td>
<td>Ground</td>
<td>Common Ground</td>
</tr>
<tr>
<td>V5</td>
<td>Voltage Output</td>
<td>Logic Level 5V</td>
</tr>
<tr>
<td>A0</td>
<td>Analog</td>
<td>Ultrasonic Range Finder</td>
</tr>
<tr>
<td>D13</td>
<td>Digital</td>
<td>Ultrasonic Range Finder</td>
</tr>
<tr>
<td>D2</td>
<td>Digital</td>
<td>Motor Controller Enable</td>
</tr>
<tr>
<td>D3</td>
<td>Digital</td>
<td>Motor Controller PWM Enable</td>
</tr>
<tr>
<td>D4</td>
<td>Digital</td>
<td>Motor Controller Enable</td>
</tr>
<tr>
<td>D5</td>
<td>Digital</td>
<td>Motor Controller PWM Enable</td>
</tr>
<tr>
<td>D6</td>
<td>Digital</td>
<td>Motor Controller PWM Enable</td>
</tr>
<tr>
<td>D7</td>
<td>Digital</td>
<td>Motor Controller Enable</td>
</tr>
<tr>
<td>D8</td>
<td>Digital</td>
<td>Motor Controller Enable</td>
</tr>
<tr>
<td>D9</td>
<td>Digital</td>
<td>Motor Controller Enable</td>
</tr>
<tr>
<td>D10</td>
<td>Digital</td>
<td>Motor Controller Enable</td>
</tr>
<tr>
<td>D11</td>
<td>Digital</td>
<td>Ultrasonic Range Finder</td>
</tr>
<tr>
<td>D12</td>
<td>Digital</td>
<td>Ultrasonic Range Finder</td>
</tr>
<tr>
<td>USB</td>
<td>USB</td>
<td>Pixycam</td>
</tr>
</tbody>
</table>

2.2 Sensors
Each robot module will have various sensors to distinguish objects in the environment.

2.2.1 Image sensing
It has been confirmed that an image sensor will be used for the robot. The image sensor that will be used is the Pixy Cmucam5. It will communicate with the Bluno Nano using the mini-USB port. Using the USB results in a faster transfer rate compared to the TX/RX UART interface.
2.2.2 Ultrasonic sensing
It has been confirmed that ultrasonic sensing will be used for the robot. There will be two HC-SR04 ultrasonic rangefinders, each one on opposite sides of the module. The sensors will communicate with the Bluno Nano through two digital I/O pins. One pin is for transmitting ultrasonic sounds while the other is used for receiving them. This allows for a module to transmit a sound wave and for other modules to “listen” for it.
2.3 Motors, Battery and Wheels

2.3.1 Motor Space Calculations
Based off of an arbitrarily chosen length y and width x, the motor space calculations for the desired layout of the motors has been determined as can be seen in Figure 16 below. Geometrically, the total length of the motors (excluding the wheels / shaft), has been determined to be as follows (x being the width, and y being the length):

\[
\frac{3x + \sqrt{3}y}{2} \leq 3.75
\]

The motor that we have decided to use has width x of 1 inch, and length y of 0.8 inches, making the overall length equal to approximately 2.06 inches, which leaves plenty of room to add the wheels (approximately 1.7 inches).

![Figure 6: Motor space calculations](image)

2.3.2 Motor and Wheel
It has been geometrically determined that the axial width of the motor will play a very important part in determining the wheel to use. Using the RB-Pol-202 brushed DC motor, the wheel that we will use has to have an axiom width that will not interfere with the PIXY cam or the walls, and be tangential to a circle that has a center in the center of the cube. With a radius of 1.5 inches, the motors would have plenty of room.

The wheel that has been determined to be used is the FXA108 50mm Wheel, which is a single-row omniwheel. It has a diameter of 50mm and a width of about 16mm which would satisfy spatial requirements.

In the conceptual design phase of this project, the idea of having two wheels with two ball-bearing casters came out with the highest weighted rating and was initially determined to be implemented in each robot module. However, the idea was reconsidered because there was a major disadvantage that was not fully taken into consideration: the robot would not be able to move perpendicular to the direction of the wheels. While it was realized at the time that the
modules would not be able to move perpendicular to the wheels, it was not realized that it would be significantly more difficult to have the modules connect on sides that are perpendicular to the wheels. This would significantly decrease the modularity aspect of the modules.

2.3.3 Battery
The power source of the robot needs to be rechargeable and provide sufficient voltage and current to all of the components in the module. The battery that has been determined to be used is the EBL 18650 Lithium-Ion. It has 3.7 voltage output with an mAh rating of 3000. Only two batteries would be required to meet the 7 voltage minimum to power the Bluno Nano. The battery has no memory effect and has overcharge protection.
2.3.4 Power
Table 2 below shows the current draw and voltage required for each component under certain conditions as well as if the components have an internal voltage regulator. It also shows the capability of the batteries and the maximum possible current draw under the most stressful conditions. The max possible current is based off of allowable current draw from the I/O pins on the Bluno Nano used in our design along with other sources of current draw indicated in the datasheet, all three motors in stall condition, the Pixycam and the current from running the two range finders off of the 5V output from the Bluno Nano. The L298N is not included in the sum of current draw since the logical aspect is part of the Bluno Nano output and the potential 2A draw is based off of the motor draw. If the module drew the maximum possible current, it would be able to run continuously for an hour.

Table 2: Power specifications

<table>
<thead>
<tr>
<th>Component</th>
<th>Current</th>
<th>Voltage Required*</th>
<th>Internal Voltage Regulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluno Nano</td>
<td>40mA (per I/O pin)</td>
<td>7V~12V</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>(233mA Total)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brushed DC Motor 6V, 11500rpm</td>
<td>70mA (free)</td>
<td>6V</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>800mA (stall)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L298N Dual H Bridge DC Stepper Motor Controller Module</td>
<td>0-36mA (logical)</td>
<td>5V-35V</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>2A (drive)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pixy CMUcam5</td>
<td>140mA (typical)</td>
<td>6V-10V</td>
<td>Y</td>
</tr>
<tr>
<td>HC-SR04 Ultrasonic Range Finder</td>
<td>15mA</td>
<td>5V</td>
<td>N</td>
</tr>
<tr>
<td>EBL 18650 Li-ion Rechargeable Batteries</td>
<td>3000mA</td>
<td>7.4V</td>
<td>N/A</td>
</tr>
<tr>
<td>Max Possible Current Out:</td>
<td>2886mA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Stated voltages are based off of an unregulated power supply

2.3.5 Magnets
Each module will have N35 Grade Neodymium magnets in each corner of each side. This approach allows the modules to effectively snap into place when a side comes in contact with the other. When two modules are ready to disconnect from each other, they will pull apart with a force greater than that of the pull force of the magnets.

Figure 9: Layout of the magnets on each side of the cube
2.4 Physical Build

The shell of the robot module will be printed using the MakerBot Replicator 2 3D printer. To make configuring the physical aspects of the robot easier, each module will have a wall that can easily detach from the base using screws. All other walls will be glued to the base. Printing the walls separately makes it easier to fabricate the module since 3D printers typically cannot print at a 90 degree angle horizontal from a vertical surface.

![Robot module with all walls attached](image)

**Figure 10:** Robot module with all walls attached

The base of the module will have a raised platform to allow the motors and wheels to lift the module slightly off the ground. A small cube will be in two of the bottom corners of the module to hold a screw used for attaching or detaching the detachable wall. A multitude of platforms are used to hold the sensors, breadboards, battery holder and the motor controllers.
Due to size constraints, there will be small sections cut out from the walls near the wheels. These will allow just enough room for the wheels to fit inside the module. This cutout will not be visible from the outside of the module and will not affect the functionality of the robot. There are small ridges printed on 2 of the walls (1 of which is visible in Figure 20, upper left wall), to hold the lid (Figure A-6, Figure 19) into place, and a small tab has been added to the lid to allow ease of disassembly.

Figure 11: Internal view of the module with all components
Figure 12: Internal view of module showing components toward the back of the cube
Figure 13: Robot module with walls set to 50% transparency
Section III: Revisions of Final Implemented Design
3.1 Motors and Motor Controller

3.1.1 Motors
Prior to full implementation of the RB-Pol-202 brushed DC motor it was determined that the motor would need to be geared down in order to have enough torque to move the modules efficiently at the low speeds needed. It was also determined that a set screw would not be adequate at the torque for the required amount of times on a smooth shaft so an alternate motor was found to resolve these issues.

The alternate motor was a 298:1 Micro Metal Gearmotor which had a similar electrical power rating and D-shaft while using a geared system to turn the speed down to 45 max RPM with a rated torque of 40 oz-in.

![Figure 14: 298:1 Micro Metal Gearmotor](image)

3.1.2 Motor Controllers
Prior to full implementation of the L298N dual H bridge DC stepper motor controller module it was determined that this controller module’s size and shape were primarily responsible for increasing the height of the module.

It was found that the TBB6612FNG dual motor driver with the addition of inverters and circuit infrastructure, could be more form fitting to the module and resolve the module’s shape into a cube.

![Figure 15: TBB6612FNG dual motor driver](image)
3.2 Magnets and Orientation

Prior to full implementation of the design of using wide square neodymium magnets in a square formation within the modules to foster inter-module connection, it was determined that narrow cylindrical neodymium magnets in a diamond formation could be used on the surface of modules to enable connection with a previously designed robot outside the scope of this project.

It was found that even with the smaller surface area of the new magnets they required more force to disengage than the motor system could provide so it was determined that only a single magnet should be present on each connection surface of the modules.
Figure 17: Module Design with Revisions
Section IV: Construction Process
4.1 Module Shell

The module shell was printed using the Rostock Max 3D printer. Each surface of each module was printed individually to accommodate the printing limitations of the 3D printer. Four modules were printed using this method, each using a different color filament with a distinct color signature. The four side surfaces were all put together using a two-part epoxy and the motor system was attached to the module's bottom surface. Once the motor system was attached, the bottom surface was put together with the side surfaces and then the top surface of the module could rest on ridges on the inside of the side surfaces.

![Rostock Max 3D printer printing sides of modules](image)

**Figure 18:** Rostock Max 3D printer printing sides of modules

4.2 Sensors and Magnets

The sensors were installed and fixed to the inner surface of module using epoxy and the magnets were placed in holes on each side surface of the modules and then epoxied in. This construction prevented shifting of components as well as preventing the PixyCam from losing focus during regular use of the module. Figure 19 below shows the inside of the module before the wiring and installation of the wheels.
4.3 Wheel and Motor Installation

The wheels were not originally meant to fit the gearmotors. Hex-shaped shaft adapters were used to make the motor shaft thick enough to be force fit into the wheel. The center of each wheel was drilled out with a half inch drill bit in order for the size of the shaft and wheel center to match up. A drill press was used to force each adapted shaft into the wheel until the Allen screws were just above the surface. The screws were then tightened around the shaft of the motor, and the adapters were forced the rest of the way into the wheel by manually forcing the turning of a nut on a bolt attached to the end of the adapter, with a washer against the bottom of the wheel to force it in place. After the wheels were constructed, each wheel was installed into the base of the module using Philips screws and metal plates as shown below in Figure 21.
Figure 20: Individual wheel with motor

Figure 21: All three wheels installed in base of module using washers
4.4 Circuit Construction

The primary circuit infrastructure was three breadboards within each module. One breadboard seated the Bluno Nano, one breadboard seated the motor controllers, and one was used for component connections. Male to female wires were used to connect the PixyCam and ultrasonic sensors to the system, and male to male wires were used between the breadboards to connect other components. Certain connections such as the motor leads and series battery connection were soldered connections. A switch was soldered and wired into the robot separating the battery connection from the system.

![Image of internal robot wiring](image)

**Figure 22**: Internal robot wiring

4.5 Pixy Camera Implementation

The PixyCam came with built-in image processing that provided the module with almost out-of-the-box functionality for the module. Before installment into the modules, each camera had to be focused and color signatures of each differently colored module loaded into the camera so the PixyCam could note when the modules were located using the camera. Furthermore, each camera had to be configured to transmit object data via UART protocol. All camera configuration was done using the open-source software PixyMon.
4.6 Bluno Nano Programming

The Bluno Nano can be programmed by inserting a mini-USB cable into the Bluno Nano and uploading code using the Arduino IDE shown below in Figure 24. Several digital pins from the Bluno Nano are used for interacting with the robot’s sensors and motor drivers. Four of the digital pins (D2-D5) are used for both of the ultrasonic sensors’ TRIG and ECHO pins. Three pins (D6-D8) are used to control the direction of each motor, and three pins (D9-D11) are used to control the PWM of the motor driver, effectively controlling how fast the motor should spin. Two of the digital pins (D12 and D13) are used for the camera’s UART transmission. The SoftwareSerial library from Arduino uses these two pins to emulate the UART protocol so that the Bluno Nano can receive data from the camera and process it depending on the Nano’s program. D12 is used for receiving the data, and D13 is not actually used but must be reserved when using the SoftwareSerial library in one of the robot module’s Bluno Nano.
4.7 Final Design

Figures 25 and 26 below are photographs of the final constructed design. The robots were assembled in order starting with the red one, then the green one, then pink, then blue.
Figure 26: Pink and Blue modules
Section V: Testing
5.1 Test Plan

5.1.1 Mobility
- Each individual module must be able to turn and move in any direction on a flat surface. A quarter turn and foot long unilateral movement should take 5 seconds max.
- Each module should be able to make all movements in tandem with up to three additional modules attached.

5.1.2 Battery Life
- The modules should be sufficiently powered by the batteries for a minimum of 45 minutes with a regular running load.

5.1.3 Wireless Connectivity
- The modules should be able to communicate with the control hub from a distance of up to 30ft.

5.1.4 Durability
- The modules should be resistant to non-superficial damage from full running speed contact with an obstacle or other module.

5.1.5 Module Connectivity
- Modules must be able to connect and disconnect at a max rate of 5 seconds per module dis/connection once the proper command is given.
- Modules must be able to assemble themselves when spaced within a circle of at least 5 feet in diameter.
- Modules must be able to form 3 distinct configurations based on color and shape.

5.1.6 Sensor Testing
- The ultrasonic sensors on the modules must be able to accurately detect distance from an object with a low resolution
- The Pixy CMUcam5 must be able to detect objects in its field-of-view based on stored color signatures.
5.2 Testing Results

5.2.1 Success

- Each module was able to turn and travel within the allotted amount of time. When connected, modules could move in tandem.
- Modules displayed the ability to continuously move using all three motors for a time greater than 45 minutes. In addition, each module was shown to work intermittently over several days without a noticeable change in performance.
- Modules sustained no non-superficial damage from regular operation or from occasional bumps during testing.
- Modules could disconnect from each other almost instantaneously in a controlled environment once both modules that were connected received the command to disconnect.
- Modules could assemble themselves into more than three different configurations.
- The ultrasonic sensors could accurately measure distances between objects as far as 70cm with a resolution of +/- 2mm. This was based off of sample source code found from an Internet article, and results will vary depending on the code.
- The Pixy CMUcam5 is able to accurately detect objects in its field-of-view based on color signature and report data back to the Bluno Nano for processing.
- The Bluno Nano was able to relay information such as camera data and ultrasonic sensor distance data to a computer via Bluetooth serial communication. The program PuTTY was able to display the camera data (object’s width, height, x and y locations relative to the camera’s field-of-view, etc.) on the screen and also provided the capability to either stop reading camera data or start reading it by sending a keyboard command to the Bluno Nano. The Arduino IDE’s serial monitor was used to display the ultrasonic sensor data in a separate test.

Figure 27: After the camera of the blue module detects the pink one, it will move toward it.
5.2.2 Areas for Improvement

- Modules were not installed with appropriate programming to check that tandem movement could be done efficiently.
- The wireless connectivity of the modules was only verified up to 3 feet.
- Module disconnection could only be achieved using an alternate magnet configuration where the magnet location of one of the disconnecting modules was changed to be on the inside of the module wall rather than the outside.
Section VI: Cost Analysis
6.1 Cost Analysis

The costs for all of the used components are shown below in Table 3. The 10KΩ resistors used for the inverters are not included, as they are self-provided by the team. A bill was not provided for the 3D robot filament, therefore an estimate based off of a previous project was made. The most expensive component of the module is the Pixy CMUcam5 due to its powerful image-processing capabilities and high programmability. After accounting for all of the components used in the robot, the total cost is $865.05. This was over $100 under budget in which the excess could be used for any future repairs or upgrades to the robotic system.

Table 3: Bill of Materials for All Four Robot Modules

<table>
<thead>
<tr>
<th>Category</th>
<th>Object Name</th>
<th>Cost Per Item</th>
<th>Quantity Ordered</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robotics</td>
<td>Bluno Nano</td>
<td>$33.55</td>
<td>4</td>
<td>$134.20</td>
</tr>
<tr>
<td></td>
<td>Pixy CMUcam5</td>
<td>$69.00</td>
<td>4</td>
<td>$276.00</td>
</tr>
<tr>
<td></td>
<td>HC-SR04 Ultrasonic Range Finder</td>
<td>$2.50</td>
<td>8</td>
<td>$20.00</td>
</tr>
<tr>
<td></td>
<td>USE BLE-Link</td>
<td>$9.90</td>
<td>4</td>
<td>$39.60</td>
</tr>
<tr>
<td></td>
<td>HEX INVERTER (O.C.) DIP-14</td>
<td>$1.39</td>
<td>4</td>
<td>$5.56</td>
</tr>
<tr>
<td>Power</td>
<td>BK-18650-PC4-ND Battery Holder</td>
<td>$4.48</td>
<td>4</td>
<td>$17.92</td>
</tr>
<tr>
<td></td>
<td>BRC 4200mAh 18650 Battery (4pc)</td>
<td>$7.59</td>
<td>2</td>
<td>$15.18</td>
</tr>
<tr>
<td>Wheel/Motor</td>
<td>FXA108 50mm Wheel</td>
<td>$3.16</td>
<td>12</td>
<td>$37.92</td>
</tr>
<tr>
<td></td>
<td>Dual TB6612FNG Motor Driver</td>
<td>$8.95</td>
<td>8</td>
<td>$71.60</td>
</tr>
<tr>
<td></td>
<td>298:1 Micro Metal Gearmotor</td>
<td>$7.99</td>
<td>12</td>
<td>$95.88</td>
</tr>
<tr>
<td>Other</td>
<td>Breadboard PRT-12043</td>
<td>$3.95</td>
<td>12</td>
<td>$47.40</td>
</tr>
<tr>
<td></td>
<td>D44 Magnets</td>
<td>$0.54</td>
<td>80</td>
<td>$43.20</td>
</tr>
<tr>
<td></td>
<td>Makerbot Filament</td>
<td>$48.00</td>
<td>1</td>
<td>$48.00</td>
</tr>
<tr>
<td></td>
<td>Jumper Wires</td>
<td>$12.59</td>
<td>1</td>
<td>$12.59</td>
</tr>
</tbody>
</table>

Total: $865.05
Conclusion

After going through the requirements analysis, preliminary design review, detailed design generation, and prototype construction and testing, we have successfully created an autonomous self-reconfigurable modular robot system that meets the minimum project requirements. The robot is controlled via a Bluno Nano which is programmable and can communicate via wireless protocol at several feet. The power source is rechargeable and can last more than a half-hour on a single charge. The modules are equipped with two different types of sensors, can connect/disconnect, can move well on different surfaces, and can form more than three visibly different configurations.

One of the main factors that affects the modularity of the robot is how the magnets are installed. The provided magnets were deemed too powerful; as a result, there is only one magnet per side of the robot. While this makes it easier for the robot modules to separate, it only allows one magnetic polarity per side. With only one polarity per side, this reduces the number of possible configurations that the robot can form.

Overall, the modular robotic system meets the minimum requirements of the project and can be used for studying artificial intelligence and for any senior design projects that may arise in the computer science department at IPFW.
References

https://www.arduino.cc/en/Main/ArduinoBoardNano


http://www.rcmodelreviews.com/baffledbybatteries.shtml


"EBL® 18650 3.7V 3000mAh Li-ion Rechargeable Batteries, 2 Pack." EBL® 18650 3.7V 3000mAh Li-ion Rechargeable Batteries, 2 Pack. Web. 06 Apr. 2016.

Appendix

Primary Source Code Files

“Pin Numbers.h”
#define CAMERA_RX_PIN 12
#define LEFT_SENSOR_ECHO_PIN 3
#define LEFT_SENSOR_TRIG_PIN 2
#define RIGHT_SENSOR_ECHO_PIN 5
#define RIGHT_SENSOR_TRIG_PIN 4
#define BACK_MOTOR_DIRECTION_PIN 6
#define LEFT_MOTOR_DIRECTION_PIN 7
#define RIGHT_MOTOR_DIRECTION_PIN 8
#define BACK_MOTOR_PWM_PIN 9
#define LEFT_MOTOR_PWM_PIN 10
#define RIGHT_MOTOR_PWM_PIN 11

“Pixy.h”
//
// begin license header
//
// This file is part of Pixy CMUCam5 or "Pixy" for short
//
// All Pixy source code is provided under the terms of the
// GNU General Public License v2 (http://www.gnu.org/licenses/gpl-2.0.html).
// Those wishing to use Pixy source code, software and/or
// technologies under different licensing terms should contact us at
// cmucam@cs.cmu.edu. Such licensing terms are available for
// all portions of the Pixy codebase presented here.
//
// end license header
//
// This file is for defining the SPI-related classes. It's called Pixy.h instead
// of Pixy_SPI.h because it's the default/recommended communication method
// with Arduino. This class assumes you are using the ICSP connector to talk to
// Pixy from your Arduino. For more information go to:
//
// http://cmucam.org/projects/cmucam5/wiki/Hooking_up_Pixy_to_a_Microcontroller_(like_an_Arduino)
//
#endif
#define PIXIY_H
#define PIXIY_H

#include "TPixy.h"
#include "SPI.h"

#define PIXY_SYNC_BYTE 0x5a
#define PIXY_SYNC_BYTE_DATA 0x5b
#define PIXY_BUF_SIZE 16

template <class BufType> struct CircularQ
{
    CircularQ()
    {
        len = 0;
        writeIndex = 0;
        readIndex = 0;
    }

    bool read(BufType *c)
    {
        if (len)
        {
            *c = buf[readIndex++];
            len--;
            if (readIndex==PIXY_BUF_SIZE)
            {
                readIndex = 0;
                return true;
            }
            else
            {
                return false;
            }
        }
    }

    uint8_t freeLen()
    {
        return PIXY_BUF_SIZE-len;
    }

    bool write(BufType c)
    {
        if (freeLen()==0)
        {
            return false;
        }

        buf[writeIndex++] = c;
        len++;
        if (writeIndex==PIXY_BUF_SIZE)
        {
            writeIndex = 0;
            return true;
        }
    }

    BufType buf[PIXY_BUF_SIZE];
    uint8_t len;
    uint8_t writeIndex;
    uint8_t readIndex;
};

class LinkSPI
{
public:
void init()
{
  SPI.begin();

#ifdef __SAM3X8E__
  // DUE clock divider
  SPI.setClockDivider(84);
#else
  // Default clock divider
  SPI.setClockDivider(SPI_CLOCK_DIV16);
#endif

  uint16_t getWord()
  {
    // ordering is different (big endian) because Pixy is sending 16 bits through SPI
    // instead of 2 bytes in a 16-bit word as with I2C
    uint16_t w;

    if (inQ.read(&w))
      return w;
    return getWordHw();
  }

  uint8_t getByte()
  {
    return SPI.transfer(0x00);
  }

  int8_t send(uint8_t *data, uint8_t len)
  {
    int i;

    // check to see if we have enough space in our circular queue
    if (outQ.freeLen()<len)
      return -1;

    for (i=0; i<len; i++)
      outQ.write(data[i]);
    flushSend();
    return len;
  }

  void setArg(uint16_t arg)
  {
  }

private:
uint16_t getWordHw()
{  
    // ordering is different (big endian) because Pixy is sending 16 bits through SPI  
    // instead of 2 bytes in a 16-bit word as with I2C  
    uint16_t w;  
    uint8_t c, cout = 0;  

    if (outQ.read(&cout))  
        w = SPI.transfer(PIXY_SYNC_BYTE_DATA);  
    else  
        w = SPI.transfer(PIXY_SYNC_BYTE);  

    w <<= 8;  
    c = SPI.transfer(cout);  
    w |= c;  

    return w;  
}

void flushSend()  
{  
    uint16_t w;  
    while(outQ.len)  
    {  
        w = getWordHw();  
        inQ.write(w);  
    }  
}

// we need a little circular queues for both directions  
CircularQ<uint8_t> outQ;  
CircularQ<uint16_t> inQ;  
}

typedef TPixy<LinkSPI> Pixy;

#endif

"TPixy.h"
  
  // begin license header  
  //  
  // This file is part of Pixy CMUcam5 or "Pixy" for short  
  //  
  // All Pixy source code is provided under the terms of the  
  // GNU General Public License v2 (http://www.gnu.org/licenses/gpl-2.0.html).  
  // Those wishing to use Pixy source code, software and/or  
  // technologies under different licensing terms should contact us at  
  // cmucam@cs.cmu.edu. Such licensing terms are available for  
  // all portions of the Pixy codebase presented here.  
  //
// This file is for defining the Block struct and the Pixy template class.
// (TPixy). TPixy takes a communication link as a template parameter so that
// all communication modes (SPI, I2C and UART) can share the same code.

#ifndef _TPIXY_H
#define _TPIXY_H

#include "Arduino.h"

// Communication/misc parameters
#define PIXY_INITIAL_ARAYSIZE 30
#define PIXY_MAXIMUM_ARAYSIZE 130
#define PIXY_START_WORD 0xaa55
#define PIXY_START_WORD_CC 0xaa56
#define PIXY_START_WORDX 0x55aa
#define PIXY_MAX_SIGNATURE 7
#define PIXY_DEFAULT_ARGVAL 0xffff

// Pixy x-y position values
#define PIXY_MIN_X 0L
#define PIXY_MAX_X 319L
#define PIXY_MIN_Y 0L
#define PIXY_MAX_Y 199L

// RC-servo values
#define PIXY_RCS_MIN_POS 0L
#define PIXY_RCS_MAX_POS 1000L
#define PIXY_RCS_CENTER_POS ((PIXY_RCS_MAX_POS-PIXY_RCS_MIN_POS)/2)

enum BlockType
{
    NORMAL_BLOCK,
    CC_BLOCK
};

struct Block
{
    // print block structure!
    void print()
    {
        int i, j;
        char buf[128], sig[6], d;
        bool flag;
        if (signature>PIXY_MAX_SIGNATURE) // color code! (CC)
        {
            // convert signature number to an octal string
            for (i=12, j=0, flag=false; i>=0; i/=3)  

{(signature>>i)&0x07;
    if (d>0 && (!flag))
        flag = true;
    if (flag)
        sig[j++] = d + '0';
}
sig[j] = '\0';
sprintf(buf, "CC block! sig: %s (%d decimal) x: %d y: %d width: %d height: %d angle %d\n", sig, signature, x, y, width, height, angle);
}
else // regular block. Note, angle is always zero, so no need to print
    sprintf(buf, "sig: %d x: %d y: %d width: %d height: %d\n", signature, x, y, width, height);
Serial.print(buf);
}
// print block structure!
void print_and_do_something()
{
    int i, j;
    char buf[128], sig[6], d;
    bool flag;
    if (signature>PIXY_MAX_SIGNATURE) // color code! (CC)
    {
        // convert signature number to an octal string
        for (i=12, j=0, flag=false; i>=0; i-=3)
        {
            d = (signature>>i)&0x07;
            if (d>0 && (!flag))
                flag = true;
            if (flag)
                sig[j++] = d + '0';
        }
sig[j] = '\0';
        sprintf(buf, "CC block! sig: %s (%d decimal) x: %d y: %d width: %d height: %d angle %d\n", sig, signature, x, y, width, height, angle);
    }
    else // regular block. Note, angle is always zero, so no need to print
        sprintf(buf, "sig: %d x: %d y: %d width: %d height: %d\n", signature, x, y, width, height);
    Serial.print(buf);
}
uint16_t signature;
uint16_t x;
uint16_t y;
uint16_t width;
uint16_t height;
uint16_t_angle;
};
public:
  TPixy(uint16_t arg=PIXY_DEFAULT_ARGVAL);
~TPixy();

uint16_t getBlocks(uint16_t maxBlocks=1000);
int8_t setServos(uint16_t s0, uint16_t s1);
int8_t setBrightness(uint8_t brightness);
int8_t setLED(uint8_t r, uint8_t g, uint8_t b);
void init();

Block *blocks;

private:
  boolean getStart();
  void resize();

  LinkType link;
  boolean skipStart;
  BlockType blockType;
  uint16_t blockCount;
  uint16_t blockArraySize;
};

template <class LinkType> TPixy<LinkType>::TPixy(uint16_t arg)
{
  skipStart = false;
  blockCount = 0;
  blockArraySize = PIXY_INITIAL_ARRAYSIZE;
  blocks = (Block *)malloc(sizeof(Block)*blockArraySize);
  link.setArg(arg);
}

template <class LinkType> void TPixy<LinkType>::init()
{
  link.init();
}

template <class LinkType> TPixy<LinkType>::~TPixy()
{
  free(blocks);
}

template <class LinkType> boolean TPixy<LinkType>::getStart()
{
  uint16_t w, lastw;

  lastw = 0xffff;

while (true)
{
    w = link.getWord();
    if (w==0 && lastw==0)
    {
        delayMicroseconds(10);
        return false;
    }
    else if (w==PIXY_START_WORD && lastw==PIXY_START_WORD)
    {
        blockType = NORMAL_BLOCK;
        return true;
    }
    else if (w==PIXY_START_WORD_CC && lastw==PIXY_START_WORD)
    {
        blockType = CC_BLOCK;
        return true;
    }
    else if (w==PIXY_START_WORDX)
    {
        Serial.println("reorder");
        link.getByte(); // resync
        lastw = w;
    }
}

template <class LinkType> void TPixy<LinkType>::resize()
{
    blockArraySize += PIXY_INITIAL_ARRAYSIZE;
    blocks = (Block *)realloc(blocks, sizeof(Block)*blockArraySize);
}

template <class LinkType> uint16_t TPixy<LinkType>::getBlocks(uint16_t maxBlocks)
{
    uint8_t i;
    uint16_t w, checksum, sum;
    Block *block;

    if (!skipStart)
    {
        if (getStart()==false)
            return 0;
    }
    else
        skipStart = false;

    for(blockCount=0; blockCount<maxBlocks && blockCount<PIXY_MAXIMUM_ARRAYSIZE;)
    {
        checksum = link.getWord();
        if (checksum==PIXY_START_WORD) // we've reached the beginning of the next frame
        {
            block = new Block();
            blockArraySize += 2;
            blocks = (Block *)realloc(blocks, sizeof(Block)*blockArraySize);
            block->type = NORMAL_BLOCK;
            block->data = 0;
            block->next = NULL;
            blockCount++;
        }
    }
}
skipStart = true;
    blockType = NORMAL_BLOCK;
    //Serial.println("skip");
    return blockCount;
}
else if (checksum==PIXY_START_WORD_CC)
{
    skipStart = true;
    blockType = CC_BLOCK;
    return blockCount;
}
else if (checksum==0)
    return blockCount;

if (blockCount>blockArraySize)
    resize();

block = blocks + blockCount;

for (i=0, sum=0; i<sizeof(Block)/sizeof(uint16_t); i++)
{
    if (blockType==NORMAL_BLOCK && i>=5) // skip
    {
        block->angle = 0;
        break;
    }
    w = link.getWord();
    sum += w;
    *((uint16_t *)block + i) = w;
}

if (checksum==sum)
    blockCount++;
else
    Serial.println("cs error");

w = link.getWord();
if (w==PIXY_START_WORD)
    blockType = NORMAL_BLOCK;
else if (w==PIXY_START_WORD_CC)
    blockType = CC_BLOCK;
else
    return blockCount;
}
}

template <class LinkType> int8_t TPixy<LinkType>::setServos(uint16_t s0, uint16_t s1)
{
    uint8_t outBuf[6];

    outBuf[0] = 0x00;
outBuf[1] = 0xff;
*(uint16_t *)(outBuf + 2) = s0;
*(uint16_t *)(outBuf + 4) = s1;

return link.send(outBuf, 6);
}

template <class LinkType> int8_t TPixy<LinkType>::setBrightness(uint8_t brightness)
{
    uint8_t outBuf[3];

    outBuf[0] = 0x00;
    outBuf[1] = 0xfe;
    outBuf[2] = brightness;

    return link.send(outBuf, 3);
}

template <class LinkType> int8_t TPixy<LinkType>::setLED(uint8_t r, uint8_t g, uint8_t b)
{
    uint8_t outBuf[5];

    outBuf[0] = 0x00;
    outBuf[1] = 0xfd;
    outBuf[2] = r;
    outBuf[3] = g;
    outBuf[4] = b;

    return link.send(outBuf, 5);
}

#endif

"PixyUART.h"

//
// begin license header
//
// This file is part of Pixy CMUcam5 or "Pixy" for short
//
// All Pixy source code is provided under the terms of the
// GNU General Public License v2 (http://www.gnu.org/licenses/gpl-2.0.html).
// Those wishing to use Pixy source code, software and/or
// technologies under different licensing terms should contact us at
// cmucam@cs.cmu.edu. Such licensing terms are available for
// all portions of the Pixy codebase presented here.
//
// end license header
//
// This file is for defining the link class for UART communications.
//
#ifndef _PIXYUART_H
#define _PIXYUART_H

#include "TPixy.h"
#include "Arduino.h"
#include <SoftwareSerial.h>

class LinkUART
{
public:
  SoftwareSerial pixySerial(CAMERA_RX_PIN, 13);
  void init()
  {
    pixySerial.begin(9600);
  }
  void setArg(uint16_t arg)
  {
  }
  uint16_t getWord()
  {
    int16_t u, v;
    
    while(1)
    {
      u = pixySerial.read();
      if (u>=0)
        break;
    }
    while(1)
    {
      v = pixySerial.read();
      if (v>=0)
        break;
    }
    v <<= 8;
    v |= u&0xff;
    return v;
  }
  uint8_t getByte()
  {
    int16_t u;
    
    while(1)
    {
      u = pixySerial.read();
      if (u>=0)
        break;
    }
    return (uint8_t)u;
  }
  int8_t send(uint8_t *data, uint8_t len)
  {
  }
};
typedef TPixy<LinkUART> PixyUART;

Source Code to Test Motor Drivers

"Motordriverexample.ino"
#define ControlPin1 6
#define ControlPin2 7
#define ControlPin3 8
#define PWMpin1 9
#define PWMpin2 10
#define PWMpin3 11

void setup() {
  // put your setup code here, to run once:
  pinMode(PWMpin1, OUTPUT); // sets the pin as output
  pinMode(PWMpin2, OUTPUT); // sets the pin as output
  pinMode(PWMpin3, OUTPUT); // sets the pin as output
  pinMode(ControlPin1, OUTPUT); // sets the pin as output
  pinMode(ControlPin2, OUTPUT); // sets the pin as output
  pinMode(ControlPin3, OUTPUT); // sets the pin as output
  delay(5000);
}

void loop() {
  // put your main code here, to run repeatedly:
  analogWrite(PWMpin1, 150);
  analogWrite(PWMpin2, 150);
  analogWrite(PWMpin3, 150);
  digitalWrite(ControlPin1, LOW);
  digitalWrite(ControlPin2, LOW);
  digitalWrite(ControlPin3, LOW);
  delay(1000);
  digitalWrite(ControlPin1, HIGH);
  digitalWrite(ControlPin2, HIGH);
  digitalWrite(ControlPin3, HIGH);
  delay(1000);
}

Source Code to Test BLE Connectivity and Pixy CMUcam5

"testing_BLE_command.ino"
//
This sketch is like hello_world but uses UART communications. If you’re not sure what UART is, run the hello_world sketch!

Note, the default baudrate for Pixy’s UART communications is 19200. Given the slow datarate and Arduino’s shallow serial FIFO, this sketch sometimes gets checksum errors, when more than 1 block is present. This is because printing more than 1 object block to the serial console (as this sketch does) causes the Arduino’s serial FIFO to overrun, which leads to communication errors.

#include "PixyUART.h"
#include "Pin Numbers.h"

PixyUART pixy;

void setup()
{
    Serial.begin(9600); // 9600 baud for the serial "console" (not for the UART connected to Pixy)
    Serial.print("Starting...
");

    pinMode(BACK_MOTOR_PWM_PIN, OUTPUT); // sets the pin as output
    pinMode(LEFT_MOTOR_PWM_PIN, OUTPUT);  // sets the pin as output
    pinMode(RIGHT_MOTOR_PWM_PIN, OUTPUT); // sets the pin as output
    pinMode(BACK_MOTOR_DIRECTION_PIN, OUTPUT); // sets the pin as output
    pinMode(LEFT_MOTOR_DIRECTION_PIN, OUTPUT); // sets the pin as output
    pinMode(RIGHT_MOTOR_DIRECTION_PIN, OUTPUT); // sets the pin as output
    delay(3000);

    pixy.init();
    digitalWrite(LEFT_MOTOR_DIRECTION_PIN, HIGH);
    digitalWrite(BACK_MOTOR_DIRECTION_PIN, HIGH);
    digitalWrite(RIGHT_MOTOR_DIRECTION_PIN, HIGH);
    analogWrite(LEFT_MOTOR_PWM_PIN, 0);
    analogWrite(BACK_MOTOR_PWM_PIN, 0);
    analogWrite(RIGHT_MOTOR_PWM_PIN, 0);
    Serial.print("Ready!
");
}
bool start = false;
void loop()
{
    static int i = 0;
    int j;
    uint16_t blocks;
    char buf[32];
    char command;
    if (Serial.available() > 0)
    {
        command = Serial.read();
        if (command == 's')
        {
            start = true;
            Serial.println("Starting robot");
            analogWrite(LEFT_MOTOR_PWM_PIN, 55);
            analogWrite(BACK_MOTOR_PWM_PIN, 55);
            analogWrite(RIGHT_MOTOR_PWM_PIN, 55);
        }
        else if (command == 'd')
        {
            start = false;
            Serial.println("Stopping robot");
            analogWrite(LEFT_MOTOR_PWM_PIN, 0);
            analogWrite(BACK_MOTOR_PWM_PIN, 0);
            analogWrite(RIGHT_MOTOR_PWM_PIN, 0);
        }
    }
    if (!start)
    {
        return;
    }
    blocks = pixy.getBlocks();

    if (blocks)
    {
        i++;
        // do this (print) every 25 frames because printing every
        // frame would bog down the Arduino
        if (i%25==0)
        {
            sprintf(buf, "Detected %d:\n", blocks);
            Serial.print(buf);
            for (j=0; j<blocks; j++)
            {
                sprintf(buf, "  block %d: \", j);
                Serial.print(buf);
                pixy.blocks[j].print();
            }
        }
    }
}
Source Code to Test Ultrasonic Sensors

“Ultrasonic_test.ino”
#include "PixyUART.h"

void setup() {
  Serial.begin(9600);
  Serial.println("Starting...");
  pinMode(LEFT_SENSOR_ECHO_PIN, INPUT);
  pinMode(LEFT_SENSOR_TRIG_PIN, OUTPUT);
  pinMode(RIGHT_SENSOR_ECHO_PIN, INPUT);
  pinMode(RIGHT_SENSOR_TRIG_PIN, OUTPUT);
}

void loop() {
  long duration_left, distance_left;
  long duration_right, distance_right;

  delayMicroseconds(2); // Added this line
  digitalWrite(LEFT_SENSOR_TRIG_PIN, HIGH);
  delayMicroseconds(10); // Added this line
  digitalWrite(LEFT_SENSOR_TRIG_PIN, LOW);

  duration_left = pulseIn(LEFT_SENSOR_ECHO_PIN, HIGH);
  distance_left = (duration_left/2) / 29.1;

  delayMicroseconds(2); // Added this line
  digitalWrite(RIGHT_SENSOR_TRIG_PIN, HIGH);
  delayMicroseconds(10); // Added this line
  digitalWrite(RIGHT_SENSOR_TRIG_PIN, LOW);

  duration_right = pulseIn(RIGHT_SENSOR_ECHO_PIN, HIGH);
  distance_right = (duration_right/2) / 29.1;

  if (distance_left >= 200 || distance_left <= 0){
    Serial.println("Left out of range");
  }
  else {
    Serial.print(distance_left);
    Serial.println(“ cm”);
  }

  if (distance_right >= 200 || distance_right <= 0){
    Serial.println("Right out of range");
  }
  else {
    Serial.print(distance_right);
    Serial.println(“ cm”);
  }
}
} 
delay(1000); 
}