

gRAPI: An indoor four-wheeled robot API for a gumstix connex™

by

Michael Simmons

Abstract

The goal of this project was to develop gRAPI, an application programming interface (API) for a gumstix connex™ motherboard in order to control an indoor four-wheeled robot. The robot platform used was the Traxxas™ E-Maxx RC electric monster truck. gRAPI was developed to control four main components: servo, motor, range sensor, and compass sensor. gRAPI was subjected to a two-phase testing process to verify its correctness and functionality. The first phase tests each component with its corresponding code. The second phase tests multiple components together to ensure proper functionality of each component within a single program. gRAPI successfully passed all tests for both phases of testing. With the development of gRAPI for a gumstix connex™ motherboard, more advanced robotics software for mapping and localization, vision, and other algorithms can be implemented.

List of Figures

Figure

- 1.1 - ISL Hallway Hummer
- 1.2 - IRIS Minimoto electric ATV project prior to modifications
- 1.3 - Current state of IRIS Minimoto electric ATV project
- 3.1 - Traxxas™ E-Maxx electric monster truck
- 3.2 - Coding development process for the gumstix connex™
- 3.3 - Devantech SRF10 wire connection locations
- 3.4 - Devantech electric compass wire connection locations
- 3.5 - Connection location for servo, motor controller, and I2C bus
- 3.6 - gumstix connex™, tweener™, and robostix™ connected together for use on project
- 3.7 - gumstix connex™; view of processor in middle of board
- 3.8 - robostix™; view of processor on the left in the middle of the board
- 3.9 - Devantech Ultrasonic range finder sensor
- 3.10 - Devantech electric compass
- 3.11 - Side view of robot
- 3.12 - View of sensors on robot

List of Tables

Table

- 3.1 - Platform selection criteria and selection
- 3.2 - Robot controller specifications: gumstix connex™ and robostix™
- 3.3 - Sensor statistics
- 3.4 - Hardware list for project
- 3.5 - Computer specifications used for project
- 3.6 - Servo and Motor Controller Specifications
- 3.7 - Oscilloscope readings for servo and motor controller
- 3.8 - Provided programs used for testing
- 3.9 - gRAPI: class and method list
- 3.10 - Motor values in PWM signal value and percent values used for gRAPI
- 3.11 - Servo values in PWM signal values and percent values used for gRAPI
- 3.12 - Phase one test programs name and description for the four components
- 3.13 - Phase one test/trial results
- 3.14 - Phase two test programs name and description for the components tested
- 3.15 - Phase two test/trial results
- 3.16 - Problems seen from misuse of I2C bus
- 4.1 - Servo class methods
- 4.2 - Motor class methods
- 4.3 - Srf10 class methods
- 4.4 - Cmps03 class methods

Introduction

The Intelligent Systems Laboratory (ISL) at CSU, Chico facilitates the development and study of intelligent systems and autonomous robots. The main goal of the ISL is to develop search and rescue robots. The ISL was partially funded by National Science Foundation (NSF) Major Research Instrumentation (MRI)/Research in Undergraduate Institutions (RUI) grant EIA-0321385 for 2003-2006 [1]. Murphy [2] defines an intelligent robot as “a mechanical creature which can function autonomously.” According to Murphy [2], “intelligent” implies that the robot does not do things in a mindless, repetitive way and “function autonomously” means that the robot can operate, self-contained, under all reasonable conditions without requiring recourse to a human operator. As of this writing, the ISL has two search and rescue robot projects: the OCNL Hallway Hummer and the IRIS Minimoto electric ATV SAR Robot.

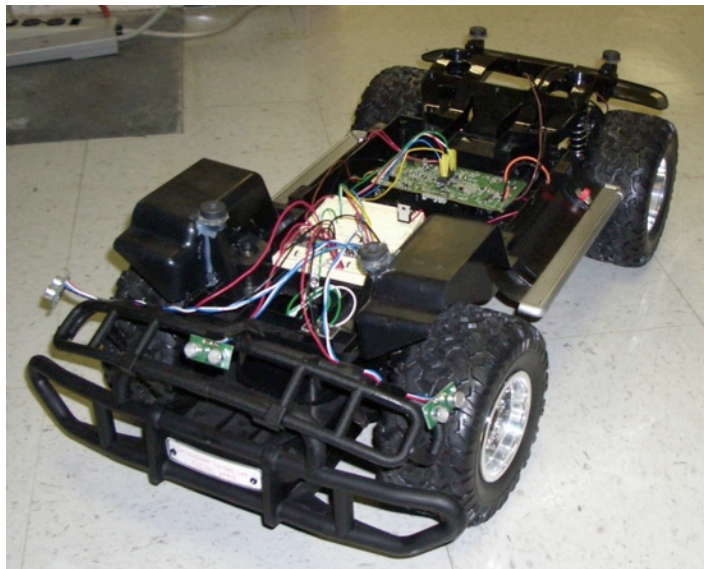


Figure 1.1 - ISL Hallway Hummer project



Figure 1.2 - IRIS Minimoto electric ATV project prior to modifications



Figure 1.3 - Current state of IRIS Minimoto electric ATV project

A picture of the ISL's Hallway Hummer project is given in Figure 1.1 and pictures of the IRIS Minimoto electric ATV project are given in Figures 1.2 and 1.3.

Purpose

The goal for ISL's intelligent robot projects is to develop functioning search and rescue robots. The OCNL Hallway Hummer is being developed and used as the prototype for the development of the IRIS Minimoto electric ATV SAR Robot. The Hallway Hummer was a remote controlled vehicle that is being converted into an autonomous robot. The remote control functionality was disabled so as to not interfere with the integration of the gumstix connexTM¹ motherboard [3] that will ultimately be the "brain" of the robot. The objective for the Hallway Hummer project as described on the project's web page [4] is for the robot to navigate the hallways of the second floor of the O'Connell Technology Center (OCNL) avoiding obstacles (walls, people walking in the hallways and stairs), store a map of the floor it navigates and take a limited set of commands. The Hallway Hummer project is still in the integration of the gumstix connexTM motherboard stage and will soon be in the development stage of the code that will control and drive the robot.

The IRIS Minimoto electric ATV SAR Robot is based on a 36V Minimoto electric ATV built for a child-sized rider, but powerful enough to carry an adult rider. The objective for the IRIS Minimoto electric ATV SAR Robot as explained on the project's

¹ gumstix connex is a trademark or registered trademark of gumstix, Inc. in the U.S. or other countries.

web page [5] is the development of an autonomous Search and Rescue (SAR) robot that can navigate through the outdoors over debris and around obstacles found in disaster areas. This robot will have two modes of operation: teleoperation mode, in which the robot will be controlled by a human operator, and a fully autonomous mode in which the robot will interact with its environment to accomplish a goal. The IRIS Minimoto electric ATV SAR Robot project moved into the development process during the last year. Specific hardware to steer and brake the Minimoto electric ATV was purchased and attached to the chassis when the development process began. Testing of the hardware still needs to be done so further items can be integrated into the system.

These two projects, despite being different in size, are closely related and compliment one another well. The Hallway Hummer, being the smaller of the two projects, allows for testing of algorithms and ideas in a controlled environment. With this control comes reassurance that everything is working correctly before introducing the robot into an unknown dynamic environment. Algorithms and ideas tested and verified on the Hallway Hummer, can then be implemented and introduced into the IRIS Minimoto electric ATV SAR Robot.

Scope

The goal of this project is to develop gRAPI, an application programming interface (API) for a gumstix connex™ motherboard [3] in order to control an indoor four-wheeled robot. With the development of gRAPI, ISL will have the needed control software for the Hallway Hummer project they are currently developing. ISL can continue research and development on the other goals for the Hallway Hummer project with the gRAPI code and gumstix connex™.

The gRAPI project is developed on a similar platform and with the same sensors as the Hallway Hummer project that ISL is developing. Closely developing the gRAPI project to the Hallway Hummer project was important to facilitate use of gRAPI with the Hallway Hummer with minimal, if any, modifications to the software.

gRAPI will need to be able to interact with the motor, sensors, and steering of the robot. The developed API should successfully allow for control of forward and backward movement, allow the robot to be steered to the left and right, and allow for reading input signals from the selected sensors for this project.

Limitations

The project limitations are grouped into four areas: Performance measure, Environment, Actuators, and Sensors. The performance measures are test-based evaluations of how the developed API successfully (1) controls the movement of the robot, (2) controls steering of the robot, and (3) reads information from the sensors.

Movement for the robot should be in the forward and reverse directions. The robot should be able to be steered to the left or right.

Limiting the environment to the indoors controls what the robot encounters during development and testing. Removing any dynamic variable from the environment facilitates comparison of expected results. This allows for the control of each component to be verified that it is functioning properly and correctly.

This project is limited to a four-wheeled robot with only two types of actuators: a motor controller to control the motors on the selected robot platform and a servo for steering the wheels.

This project is limited to only two types of sensors: a range and navigation sensor. The range sensor returns the distance from the nearest object. This range sensor can range a distance of six meters, but that maximum range is limited by the height the sensor is mounted on the robot. The navigation sensor gives the direction the robot is moving but the sensor can be affected by electro magnetic interference since it is reading the magnetic fields from the earth. The accuracy of the navigational sensor is limited to tenths of a degree when a reading of the sensor is given.

Test Plan

Testing of the developed API should be done in two phases. First, the developed code should be tested with each component. Then testing of multiple components in a

single program for proper functionality of each component can be done. In the first phase of testing the test programs will be simple but should confirm complete functionality of the component. The programs for the second phase of testing will be a little more complex but simple enough to ensure that each component in the test is functioning correctly.

Definition of Terms

Artificial Intelligence (AI) - The science of making machines act intelligently [2].

Robot - a collection of mechanisms, sensors and actuators with no common purpose [6].

Application Programming Interface (API) - a set of routines, protocols, and tools for building software applications [7].

controller - a device or software that tie the robot's components together into an integrated system with the flexibility to perform a variety of tasks by rapid re-programming [6].

Review of Related Literature

The articles reviewed for this chapter show the importance of a working API that controls a robot. With a functioning API for a robot, various interesting robotics problems like mapping and localization, vision, or AI algorithms can be supported for development. The area of mapping and localization was selected to illustrate the complexity inherent in robotics tasks.

Mapping and Localization

A look at the area of mapping and localization for a robot shows the importance of a working API so that the more complex goal can be achieved by the robot. Research can then be focused on the specific algorithms that are used for mapping and localization. These algorithms will use the API to control the robot and gather information about the environment, so the algorithms can successfully map and locate the robot in the environment.

In mobile intelligent robots, mapping and localization are two key parts of the robot's make-up that are continuously being looked at and improved. In recent publications the study of simultaneous localization and mapping (SLAM) has become the focus of research in this area of robotics. In the publication by Begum *et al.* [8] it mentions four issues that need to be addressed and resolved in the solution for SLAM.

The first issue that needs to be addressed is sensor uncertainty which cause errors in the generated maps by the algorithm. The second issue is correspondence problems in which the robot has taken different reading of the same location from different points in the environment. A loop closing problem is the third issue, and consists of the need of the robot to realize that it is in a cyclic environment and recognize its starting and finishing location so it can terminate its mapping of the environment. Lastly, the time complexity issue addresses the algorithm's need to work in real time with the sensor data. To support the issue on loop closure Ho and Newman's [9] work focuses solely on that issue for the SLAM solution and offered improvements to the loop closure issue.

There have been many algorithms developed trying to efficiently solve the SLAM problem. Some of these algorithms are the Extended Kalman Filter mentioned in the following publications [10, 11], the Rao-Blackwellized partial filter [12], RatSLAM [13], and different geometric SLAM algorithms [11, 14]. Each of these algorithms attempt to improve different issues that need to be addressed within the selected algorithms.

The sensors used on the robots for the different algorithms are usually visual and sometimes spatial sensors. The RatSLAM algorithm developed by Milford *et al.* [13] was implemented using a visual sensor. Another visual algorithm using a series of stored photos that are used to orient and guide the robot through its environment was developed by Remazeilles and Chaumette [15]. There are other visual algorithms that work on object recognition in the robot's environment to move the robot within its environment like that developed by Vasudevan *et al.* [16]. There are some algorithms that use only

spatial sensors in their solution to the SLAM problem one being the algorithm developed by Wang and Liu [17]. Using visual or spatial sensors have certain advantages and can add to the complexity of the algorithm developed to handle the data from the sensors.

Mapping and localization is an area of robotics where an algorithm is used to store and organize data about the robot's environment, or using the stored data about the environment as well as current data about the robot's immediate environment so the robot can figure out on its own where it is located. Also some algorithms for mapping and localization use a visual sensor to gather environment data, introducing another area of research for robotics. Having a working API that controls the robot allows for enhanced development on the more complex actions and behaviors of a robot.

Controller Design

With the progress and development of robotics, research in this field is being conducted in mapping, localization, vision, and more advanced AI algorithms. Having a solid development platform can allow for research in the areas mentioned above.

Development of a applied programming interface (API) to use to control a robot would allow for such research to happen and take place. McKerrow states that without software, a robot is a collection of mechanisms, sensors and actuators with no common purpose. Computer control programs tie these components together into an integrated system with the flexibility to perform a variety of tasks by rapid re-programming [6].

This software can be developed in two ways; functional modules like mentioned by Jones

and Flynn [18] or task-achieving behaviors rather than functions proposed by Brooks [19].

There are two approaches to developing control software when designing and development take place. The software can be written to work with specific hardware and sensors, or the software can be written to work with many if not all of the available hardware and sensor configuration. Both will end up controlling a robot or autonomous machine the size and complexity of the developed control software will be the only varying difference. An example of writing for a specific platform is the paper on a controller that used the Esterel tools by Sowmya *et al.* [20]. In their project they used the Rug Warrior Pro robot platform and developed the software that takes the object code from the Esterel tools and maps it to Interactive C code that is run directly on the Rug Warrior robot [20]. An example of developing a control system that will handle the majority of equipment for manufacturing is the paper by Hong *et al.* in the development of PC-ORC. PC-ORC stands for PC-based open robot control [21]. Hong *et al.* developed a control system that would work with equipment in the manufacturing industries. They did not select what equipment the software would control, but design the system to work with all equipment that was design according to specific standards for that field [21]. After the development of the system was finalized they tested the system with a specific item that is used in the manufacturing industry to observe the results of the developed control system.

Methodology

To develop the software to control the robot using the gumstix connex™ motherboard [3] several preliminary/preparatory steps had to be done to facilitate the development of the software. In this chapter I discuss the four steps taken to implement this project. The four steps involved are: (1) the preliminary stage of selecting hardware components and the development language; (2) the setup of the computer to compile the code that runs on the gumstix connex™; (3) hardware setup and installation; and (4) software development and testing.

Preliminary Stage

Specific details regarding the following needed to be decided upon: the robot platform; the hardware that would control the robot; and the programming language in which the software would be developed. The appropriate selection of these three items is crucial to the outcome and development of this project.

Table 3.1
Platform selection criteria and selected criteria for project

| Platform Selection Criteria | | | Selected |
|-----------------------------|---------|---------|----------|
| Environment | Indoor | Outdoor | Indoor |
| Power Source | Battery | Fuel | Battery |
| Ground Clearance | High | Low | High |

Limiting the working area of this robot to an indoor environment required that the development platform meet size and sound restrictions. Remote controlled (RC) vehicles

were looked at because they would meet the indoor restrictions placed on the project. RC vehicles can be purchased in many sizes that were small enough to allow for movement and navigation in an indoor environment. Sound levels from a RC vehicle vary from the type of motor or engine used to run the vehicle, and powering the vehicle needs to be made from selecting a fuel or battery powered vehicle (see Table 3.1). I decided to select a battery powered RC vehicle. This type of RC vehicle meets sound restrictions and would eliminate the worry of running the vehicle in a well ventilated area because of exhaust from the engine. The final requirement that was needed for the robot platform was ground clearance. The ranging sensors used on the project are limited by their distance from the floor. The higher the sensors are placed from the floor the greater the distance they can range.



Figure 3.1 - Traxxas™ E-Maxx monster truck

After looking at different types of battery powered RC vehicles the Traxxas^{TM 2} E-Maxx monster truck [22] was chosen (see Figure 3.1). This RC truck meets all the above requirements, plus parts are readily available online or from hobby stores for repair and upgradability.

Table 3.2
Robot controller specifications: gumstix connexTM and robostixTM

| Robot Controller | | |
|------------------------------------|------------------------|-----------|
| gumstix connexTM | | |
| | Processor Type | ARM |
| | Processor Speed | 400 MHz |
| | Memory | 16MB |
| robostixTM | | |
| | Processor Type | AVR |
| | Processor Speed | 16 MHz |
| | Memory | 128 Kbyte |

The gumstix connexTM motherboard [3] was chosen to control the robot (see Table 3.2). It's small size, 400MHz processor, and expansion boards allow for development of a robot using the gumstix connexTM as the controller. The expansion boards needed for the project are the tweener^{TM 3} [23], robostix^{TM 4} [24], and wifistix^{TM 5} [25]. The tweenerTM board is an expansion board with a serial port for connecting the gumstix connexTM/robostixTM combination to the computer by a serial cable for programming and

² Traxxas is a trademark or registered trademark of Traxxas.

³ tweener is a trademark or registered trademark of gumstix, Inc. in the U.S. or other countries.

⁴ robostix is a trademark or registered trademark of gumstix, Inc. in the U.S. or other countries.

⁵ wifistix is a trademark or registered trademark of gumstix, Inc. in the U.S. or other countries.

control. The robostix™ is a specifically developed board for robotics by Gumstix that has the pinout headers that are most commonly used to control a robot. The wifistix™ gives the gumstix connex™ access to a wireless router for programming and control.

Table 3.3
Sensor statistics

| | Statistics |
|----------------------------|---------------------|
| Range Sensor | 3cm - 6m range |
| | I2C interface |
| | Ultrasonic |
| | |
| Navigational Sensor | 0.0 - 359.9 degrees |
| | I2C interface |

To gather information about the robot's environment, two types of sensors were selected; the Devantech SRF10 ultrasonic range finder sensor [26] was selected to allow the program to know how close the robot was to objects or people in its environment, and the Devantech CMPS03 electronic compass sensor [27] for navigation was selected to allow the robot to use the direction it is facing to accomplish a desired task if needed (see Table 3.3).

Table 3.4
Hardware list for project

| Hardware | |
|-----------------|---|
| | Traxxas™ E-Maxx Monster Truck |
| | 4 x 6 cell 7.2 V battery for powering the robot |
| | 2 x battery chargers for the 6 cell batteries |

| | |
|--------------------------|---|
| Hardware | |
| Sensors | |
| | 3 x Devantech Ultrasonic Range Finder sensors |
| | Devantech electronic compass sensor |
| Controller | |
| | gummstix connex™ motherboard |
| | tweener™ |
| | robostix™ |
| | wifistix™ |
| | serial cable |
| | power cable |
| Lab Equipment | |
| | DC power supply |
| | multimeter |
| | oscilloscope |
| | 10" x 15" sheet of plexiglass |
| | 2 x 6" bolt and nut and lock washer |
| | 22 AWG wire for connecting hardware |
| | wire cutter and stripper |
| Development Tools | |
| | Desktop computer running Ubuntu Linux |
| | Laptop computer running Ubuntu Linux |
| | wireless router |

[Table 3.4 is a list of all the hardware used for this project.](#)

Computer Setup

Table 3.5
Computer specifications used for the project

| Computer Specifications | Desktop | Laptop |
|-------------------------|--------------|--------------|
| Processor Speed | 2250 MHz | 1700 MHz |
| Memory | 1 Gig | 512 MB |
| Operating System | Ubuntu Linux | Ubuntu Linux |

The Gumstix website has specific instructions for installing required software to compile and run code on the gumstix connex™ [28]. For this project a desktop and laptop computer running Ubuntu [29], a version of Linux, were used to compile and run the programs on the gumstix connex™ motherboard (see Table 3.5).

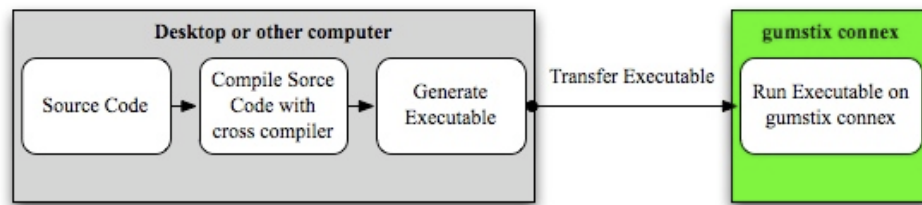


Figure 3.2 - Coding development process for the gumstix connex™

Some of the software that is installed on the computer is the kernel and libraries that are used in compiling the code that will run on the gumstix connex™. All code that is written for the gumstix connex™ is cross compiled with these libraries that work and run on the gumstix connex™. A summary of the development process for generating code to run on the gumstix connex™ motherboard is given in Figure 3.2.

Hardware setup

Wiring the sensors used for the robot was a matter of following the provided connection diagrams. The Devantech SRF10 Ultrasonic Range Finder wiring is illustrated in Figure 3.3 (SDA and SCL are data and clock line respectively).

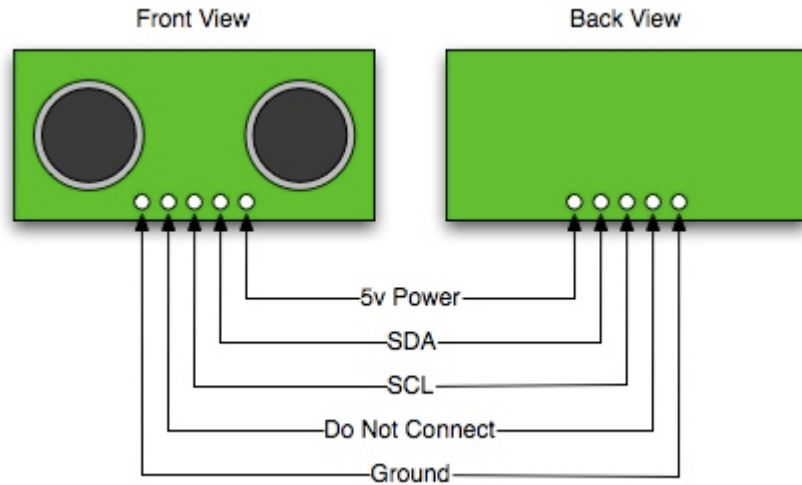


Figure 3.3 - Devantech SRF10 Ultrasonic range finder sensor wire connection locations

The Devantech electronic compass wiring is illustrated in Figure 3.4.

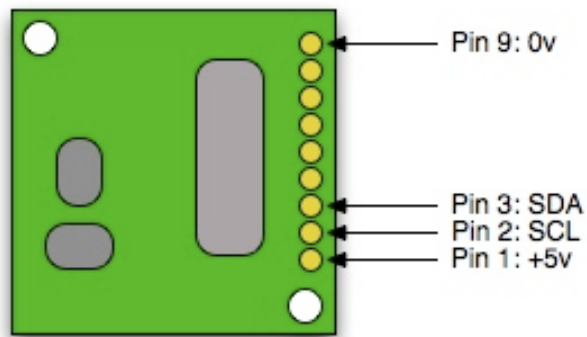


Figure 3.4 - Devantech electric compass wire connection locations for project

Table 3.6
Servo and Motor Controller Specifications

| | | |
|-------------------------|-----------------------|--------------|
| Servo | | |
| | Torque | 72 oz-in |
| | Transit time | .22 sec/60° |
| | Part Number | 2018 |
| Motor Controller | | |
| | Input Voltage | 14.4 V |
| | Directions Controlled | Forward |
| | | Reverse |
| | | Brake (Stop) |
| | Model Number | 3014 |

With the sensors wired and ready to be connected to the robot a diagnoses of the RC vehicles components was done to determine the minimum and maximum pulse width modulated (PWM) signal to control them. There are two components that will be connected to the gumstix connex™ to control movement and steering: the servo [30] for steering and the motor speed controller [31] for movement of the robot. [Table 3.6 contains some specifications on the servo and motor speed controller used.](#)

Table 3.7
Oscilloscope readings for the servo and motor controller
on the Traxxas™ E-Maxx RC truck

| Servo | Oscilloscope Reading |
|---------------------------|-----------------------------|
| Maximum Left Turn | 1 ms pulse width |
| Maximum Right Turn | 2 ms pulse width |
| Center | 1.5 ms pulse width |
| Motor Controller | |

| Servo | Oscilloscope Reading |
|-----------------------|----------------------|
| Maximum Speed Forward | 2 ms pulse width |
| Maximum Speed Reverse | 1 ms pulse width |
| Stop | 1.5 ms pulse width |

Using an oscilloscope to measure the signal that controls each device on the RC vehicle will allow for duplication of that signal by the gumstix connex™ and program that will control the robot (see [Table 3.7](#)). Connecting the RC vehicle's servo to the oscilloscope and using the remote control for the vehicle to turn the wheels would allow us to determine the minimum and maximum values to control the servo. The minimum and maximum value for the servo was 1 and the maximum value was 2ms. Centering the wheels is a value of 1.5ms. The minimum value is a maximum turn to the left, where the maximum value is a maximum turn to the right. In connecting the motor speed controller the minimum and maximum value used to control this component is the same as the servo. A value of 1.5ms is stop, a value of 1 is full speed backwards, and a value of 2ms is full speed forwards.

A platform was built out of plexiglass to hold the gumstix connex™/robostix™ and the sensors above the components of the RC vehicle. The platform is attached to the body of the RC vehicle at the same points the truck shell would be attached if the RC vehicle were used as a toy. The platform contains all of the components for this project and what make this RC vehicle become a robot.

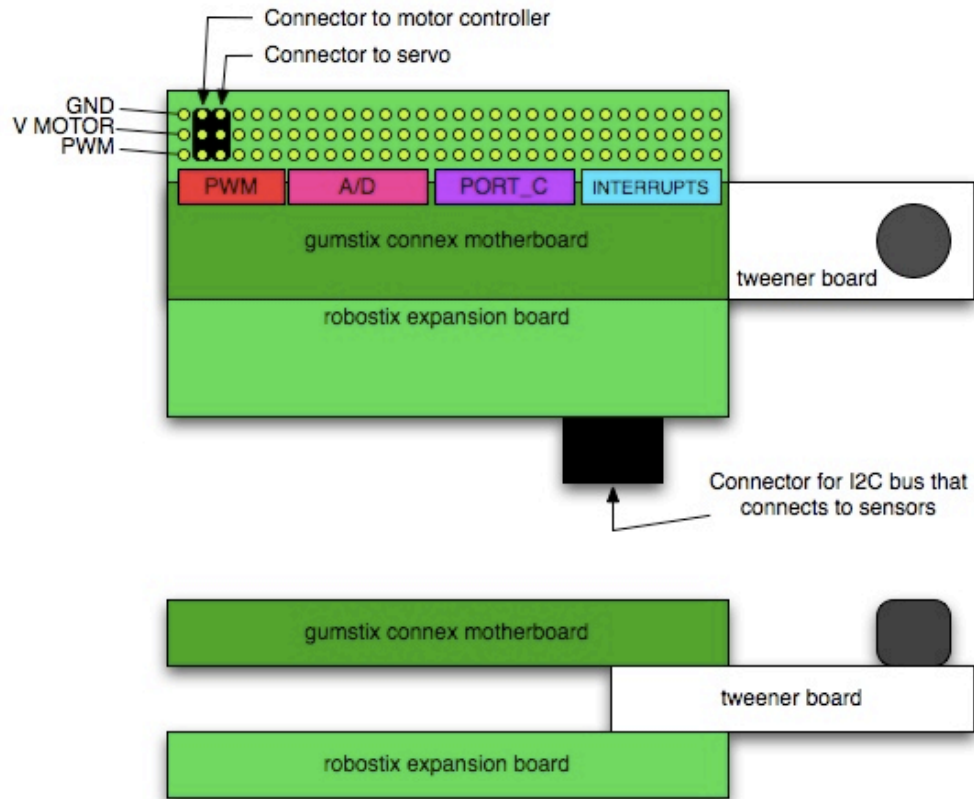


Figure 3.5 - Connection locations for servo, motor controller, and I2C bus

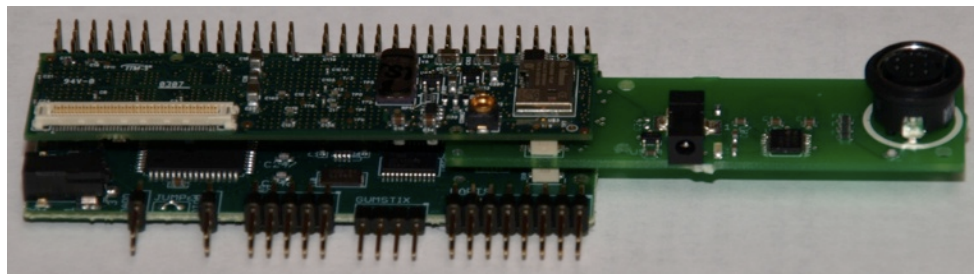


Figure 3.6 - gumstix connex™, tweener™, and robostix™ connected together for use on project

Connecting the servo, motor speed controller, and sensors to the robostix™ is a matter of connecting the leads from all of these devices to the correct pinouts on the

robostix™ (see Figures 3.5 and 3.6). The servo and motor speed controller are connected to the PWM pinouts and the sensors are connected to the I2C bus on the robostix™.

Table 3.8
Provided programs from Gumstix that were used for testing

| Program Name | Description |
|----------------|---|
| i2c.c | used to test components on i2c bus |
| Simple-Servo.c | used to test servo and motor controller |

To test the hardware setup, I ran code provided by the Gumstix website [32, 34] to ensure that all the devices are functioning and that they are connected to the right pinout on the robostix™ (see [Table 3.8](#)).

Software Setup and Development

Code used to test the sensors and gumstix connex™/robostix™ connection is available at Gumstix's website for use with their products. Sensors need to be tested to make sure that they are all functioning correctly or to know if one is malfunctioning and needs to be replaced. From the Gumstix website there is an I2C program [32] that was used to test the SRF10 range sensors and compass sensor. The code for the I2C program [32] was modified to correctly work with each sensor tested. By running this program and testing the sensors the complexity of the program developed could be greatly reduced to a few reads and writes on the I2C bus.



Figure 3.7 - gumstix connex™, view of processor in middle of board

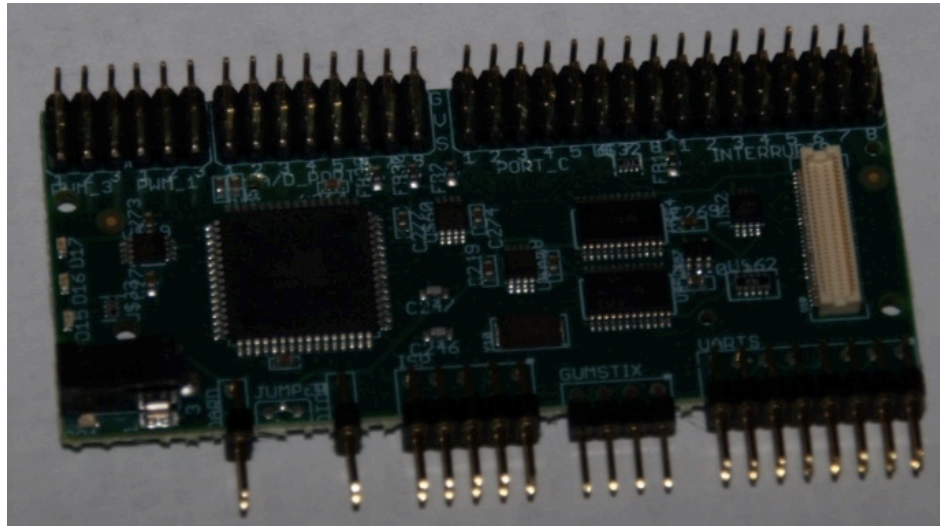


Figure 3.8 - robostix™, view of processor on the left in the middle of board

The next task was to test the gumstix connex™/robostix™ connection. The gumstix connex™ has its own processor (see Figure 3.7) and so does the robostix™ (see Figure 3.8). The gumstix connex™ processor is an ARM [3] processor and the robostix™ processor is an AVR [24]. The gumstix and robostix are programmed differently but can work together to control a robot through some software [33] on the Gumstix website. The robostix™ could run without the gumstix connex™ attached to it but would be limited to what it could do by the memory and capabilities of the processor.

Using the two together allows for more sophisticated software to be developed for robotics.

With the software installed on the gumstix connex™ and on the robostix™, communication between the two is possible and now the gumstix connex™ can control all the pinouts that are available to the robostix™. Testing the servo with provided code [34] from the Gumstix website was done and turning the wheels of the vehicle was accomplished.

To develop the code for this project an understanding of how the software between the gumstix connex™ and the robostix™ was needed. Also a closer look at the servo program and how it worked to control the turning of the wheels was conducted. An extensive tracing of the provided code [35] was done. Again like the I2C program most of the controlling of the pinouts on the robostix were done with reads and writes. After making small changes to the available code and getting the desired results, enough understanding was gained to start writing the code for this project.

The code developed to control the basic functionality of a four wheeled robot using the gumstix connex™ motherboard is written in C++, where all of the available code that has been used for testing was written in C. There are three specific areas of the developed code: vehicle movement, vehicle steering, and control of the onboard sensors.

Table 3.9
gRAPI: Class and method list

| Compass Class | |
|--|---|
| Cmps03(int handle) | compass constructor |
| duble readC() | read compass value |
| int calibrate() | used to calibrate the compass sensor |
| Motor Class | |
| Motor(int handle, int low, int high, int pwm, int delay); | motor constructor |
| void stop() | stop |
| void forward(int percent) | move forward at percent value used |
| void reverse (int percent) | move backwards at percent value used |
| int increase(int percent) | increase movement percentage used |
| int decrease(int percent) | decrease movement percentage used |
| Servo Class | |
| Servo(int handle, int low, int igh, int pwm) | servo constructor |
| void center() | center servo or wheels |
| void right(int percent) | move serve right of center percent value used |
| void left(int percent) | move servo left of center percent value used |
| int increase(int percent) | increase servo movement percentage used |
| int decrease(int percent) | decrease servo movement percentage used |
| Srf10 Class | |
| Srf10(int handle, int addr, char rt = 'c' or 'i' or 's') | SRF10 constructor |
| void ping() | ping sensor |
| int readS() | read SRF10 value |

| Compass Class | |
|--------------------------------------|--|
| void setGain(int gainValue) | set the gain value for the sensor |
| void setRange(int rangeValue) | set the range value for the sensor |
| void setResultType(char) | change sensor value type: c = centimeter, i = inches, s = microseconds |
| void changeAddress(int newAddress) | change the sensors device address |

Developing the code for this project in C++ allows a user of the code to take advantage of the concept of object oriented programming (OOP). This allows for the control functions of a motor, servo, or sensor to be packaged in objects corresponding to a class (see [Table 3.9](#)). A class is designed for each major device on the robot that needs to be controlled for the vehicle to operate. Developing the projects code with the OOP in mind would allow for easy of maintaining the code or adding additional functionality to the code that is needed in the future and reduce duplicating code to control all the devices for this project. Consider the case where two servos are needed to complete a task. Using the constructor form the servo class I would use it twice, once for Servo1 and again for Servo2. Now controlling the two servos are a matter of specifying the correct servo and then issuing the command; like Servo1.center(), this command would move the servo to its center location. There are four classes in this project; a servo class, a motor class, a SRF10 ultrasonic sensor class, and a compass class. Each of these classes have a constructor and corresponding functions to control that type of device.

Table 3.10

Values to operate the motor; PWM signal value and percent value used for gRAPI

| Motor | PWM Value | Percent Value |
|------------------------|------------------|----------------------|
| Maximum Reverse | 1000 | 100 |
| 50% Reverse | 1250 | 50 |
| Stop | 1500 | NA |
| 50% Forward | 1750 | 50 |
| Maximum Forward | 2000 | 100 |

The robots movement consists of three commands; forward, backwards, and stop. All three of these commands activate the vehicles motor speed controller through a PWM signal that is produced by the robostix™. By varying the PWM signal the speed the vehicle travels either forward or backwards is controlled (see Table 3.10). Through testing various PWM signals set values were selected to move the vehicle forward and backwards. These values were fast enough to move the vehicle in the corresponding direction required by the program, but still slow enough to ensure control and safety of others that might be around the robot. A forward value of 20 and a backwards value of 20 were selected. The last command for the movement of the vehicle was the stop command. Stopping the vehicle is a matter of issuing the stop command which actually is just setting the value to zero or the mid-point of the controllers range of values.

Table 3.11

Values to operate the servo; PWM signal value and percent value used for gRAPI

| Servo | PWM Value | Percent Value |
|-------------------------|------------------|----------------------|
| Maximum Left | 1000 | 100 |
| 50% to the Left | 1250 | 50 |
| Center | 1500 | NA |
| 50% to the Right | 1750 | 50 |

| Servo | PWM Value | Percent Value |
|---------------|-----------|---------------|
| Maximum Right | 2000 | 100 |

The robots steering consists of three commands; left, right, and center. By sending a PWM signal through the robostix™'s pins where the servo is connected the steering of the vehicle can be controlled (see Table 3.11). Centering the wheels of the vehicle is a matter of sending the mid-point value of the servo's range to the servo, which is 1500. Any value less than 1500 will turn the wheels to the left. Any value over 1500 will turn the wheels to the right. The commands for turning the wheels take an integer value that represents the percentage the wheels are to turn from center to the minimum or maximum value of the servo.

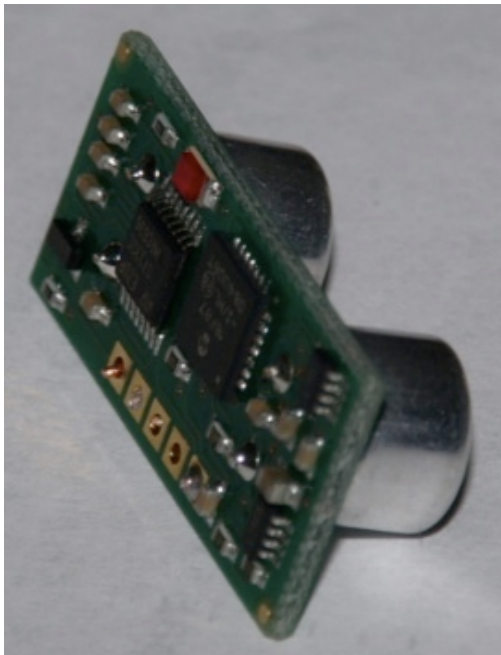


Figure 3.9 - Devantech Ultrasonic range finder sensor

Controlling of the two sensors that are used for this project is a matter of issuing a ping command and/or a read command. For the Devantech Ultrasonic Range Finder sensor (see Figure 3.9), use the ping command to cause the sensor to clear its buffer, initiate the ping, and set the value type to be stored in the buffer from the ranging of the sensor. There are three value types the sensor can report back to the user by the value stored in the buffer: inches, centimeters, or microseconds. For the project centimeters were used in the test programs for better accuracy from the sensor without having to do any further calculations on the value being returned by the sensor. The sensor is set for a 65 millisecond ranging time which is equivalent to 11 meters which is more than the maximum range of 6 meters that the sensor can range to [25]. To range something closer than its maximum ranging capabilities would require less time and the sensor can be adjusted and setup to reduce the maximum ranging capabilities of the sensor if desired. For this project the Devantech Ultrasonic Range Finder sensors were not modified, but the wait time was shortened to speed up the running of the code since the sensors were not ranging anything close to its maximum value. Once the time passed for the sensor to complete its ranging the read command is issued to read the value stored in the sensors buffer and making it available to use within the code that is running.

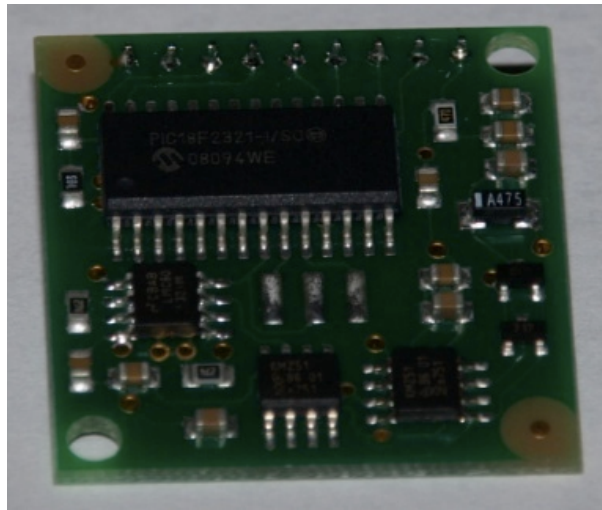


Figure 3.10 - Devantech electric compass sensor

The Devantech electronic compass [26] (see Figure 3.10) is similar to the ultrasonic range finder but only the read command is needed to read the value that the sensor has in its buffer. The compass continually updates its buffer with the current heading or direction the compass is facing. The value in the buffer is a four digit integer that represents the heading from 0.0 to 359.9 degrees with one decimal place of accuracy. In issuing the read command for the compass the program gets the current heading of the robot or direction the vehicle is facing at that moment and then can use that information within the running code. A heading of zero degrees usually is a north bound heading. The compass can also be calibrated to improve its accuracy or it can be changed to a relative north orientation to better suite the environment it will be used in.

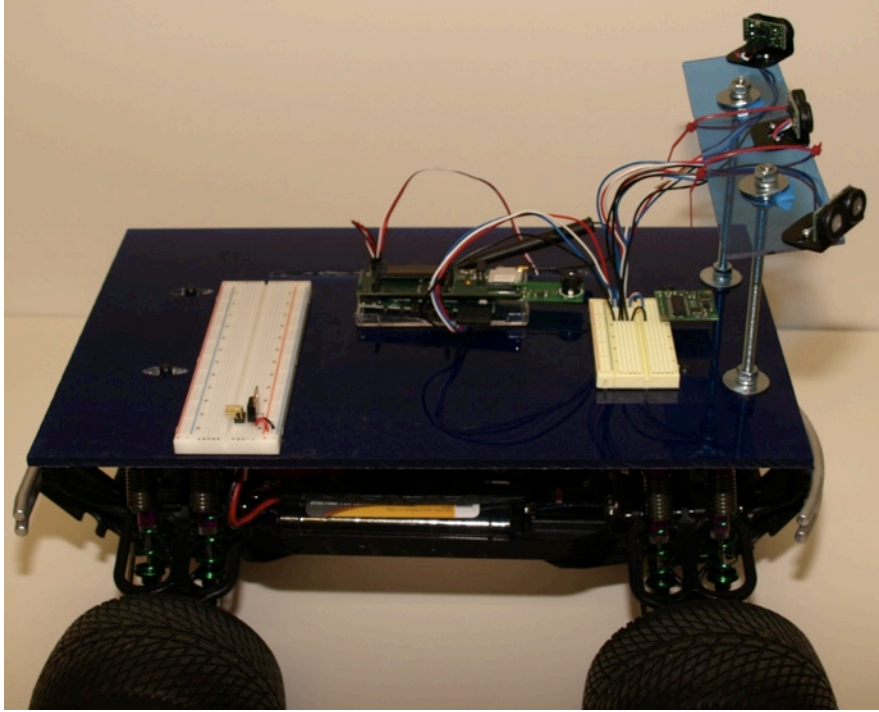


Figure 3.11 - Side view of robot

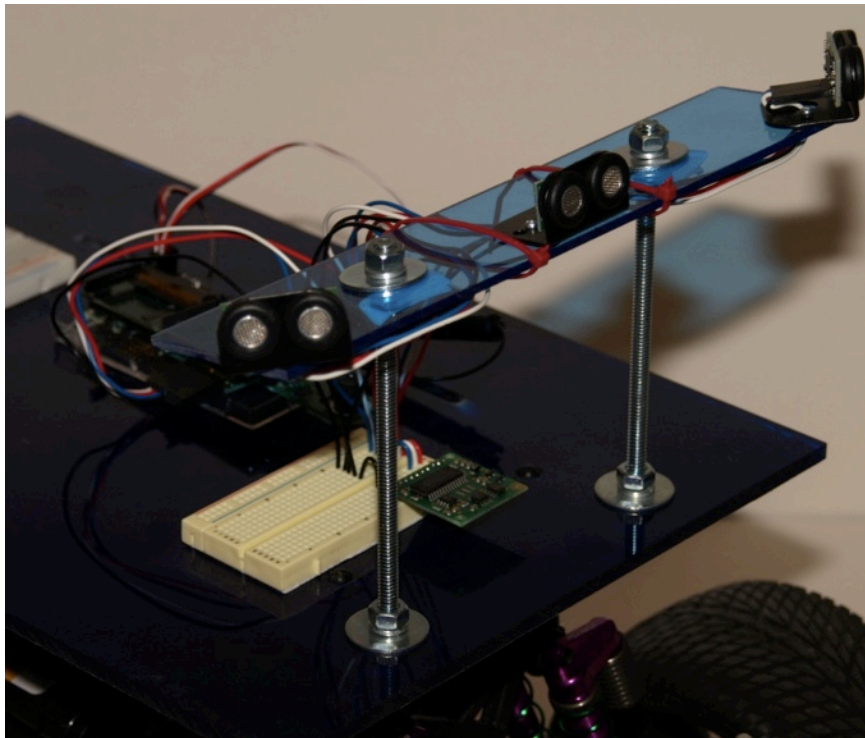


Figure 3.12 - View of sensors on robot

Pictures of the completed robot and mounted sensors are given in Figures 3.11 and 3.12.

Testing

After implementing the commands to control the robot test programs were written to test the functionality and to ensure proper coding of the developed code. Two phases of testing occurred: first, control command testing without any other control feature being tested at the same time, and then multiple control commands were tested within a single program.

Table 3.12
Phase one test programs name and description for the four components

| | Program Name | Description |
|---------------|--------------|---|
| Servo | | |
| | testServo | moved servo through complete range |
| Motor | | |
| | testMotor | moved motor forwards and backwards |
| Srf10 | | |
| | testSensor | pinged sensor and read result from buffer |
| Cmps03 | | |
| | testCompass | read result from buffer |

Table 3.13
Phase one test/trial results

| | # of times test was run | Test # | Parameters | Results |
|------------------|-------------------------|--------|-----------------------------------|---|
| testServo | 3 | 1 | left(100) - left(1) | moved wheels from max. left to center |
| | | 2 | right(100) - right(1) | moved wheels from max. left to center |
| | | 3 | left(100) - center() - right(100) | moved wheels from max. left to max. right |
| testMotor | 3 | 1 | forward(20) | moved the wheels forward |
| | | 2 | reverse(20) | moved the wheels in reverse |

| | # of times test was run | Test # | Parameters | Results |
|--------------------|-------------------------|--------|--|---|
| | | 3 | forward(20), stop(), reverse(20), stop() | moved wheels forward, stopped, reverse, stopp |
| testSensor | 2 | 1 | ping with inch, then display value | read and displayed value in inches |
| | | 2 | ping with centimeter, then display value | read and displayed value in centimeters |
| testCompass | 1 | 1 | read and displays value of current headi | read and displayed value of current heading |

In the first phase of testing (see [Table 3.12](#)) a test program was written to only test the forward, backward, and stop commands. The forward command as well as the backward command were run for a set time, 2 seconds, and then the stop command was given. In testing the steering commands another program was written to prompt the user for a value between 1100 and 1900, the minimum and maximum value of the servo. The program would initiate the starting value to 1500 which would center the wheels. The user then could enter a value at the prompt and the wheels would turn to the correct turn position. The sensors were tested with their appropriate commands with a program that loops infinitely while pinging and reading the sensors' buffer or just reading the sensors' buffer. All commands worked properly by controlling or getting readings from the appropriate component being tested (see [Table 3.13](#)).

Table 3.14
Phase two test programs name and description for the components tested

| | Program Name | Description |
|------------------------------|--------------|--|
| Motor, Servo, Compass | pointNorth | moves robot so it is heading north |
| | figure8 | moves robot in a figure a pattern |
| Motor, Servo, SRF10 | | |
| | wallHugger | follows the wall using the left sensor |

Table 3.15
Phase two test/trial results

| | # of times test was run | Test # | Results |
|-------------------|-------------------------|--------|--|
| pointNorth | 8 | 1 | Didn't go in reverse sometimes. Modified reverse code |
| | | 2 | Didn't go forward sometimes. Modified forward code |
| | | 3 | Didn't go forward or in reverse sometimes. Modified both |
| | | 4 | Moved robot to point north with only forward code running |
| | | 5 | Moved robot to point north with only reverse code running |
| | | 6 | Didn't go forward or in reverse sometimes. Looked at code |
| | | 7 | Put micro-sleeps after forward and reverse calls, worked with less probl |
| | | 8 | Put micro-sleeps after all i2c calls, worked correctly |
| figure8 | 3 | 1 | Worked correctly, but went too slow |
| | | 2 | Worked correctly, but went too fast |
| | | 3 | Worked, and found a good speed |
| wallHugger | 5 | 1 | Over corrected. Modified range code |
| | | 2 | Was to slow on correcting. Modified range code |
| | | 3 | Changed algorithm, was over correcting. Made modification to new co |
| | | 4 | Worked but was a little slow on correcting. Looked at code |
| | | 5 | Worked. Found good values for algorithm |

In phase two of testing (see [Table 3.14](#)) the developed code required programs that would test many components, if not all of the components, that need to be controlled for the robot to function properly. Three test programs were used. First, a point north program was developed to test all commands except those developed for the ultra sonic range finder sensor. A second program that was to have the robot travel in a figure eight pattern was developed, it also tested all the commands except those for the ultra sonic range finder sensor and the backwards command. The last program developed was a left wall hugger program which tested all commands except the compass commands. [All commands used in each program worked properly by controlling or getting readings from](#)

[the appropriate component being tested \(see Table 3.15\). Phase two test programs can be found in appendix C.](#)

Table 3.16
Problems seen from misuse of I2C bus

| | |
|----------------------|-------------------------|
| Problems seen | Did not go forward |
| | Did not go reverse |
| | Error messages for code |

In the second phase of testing additional things were learned about the I2C bus and how to successfully control devices on the bus. Initially, the test code for the second phase of testing did not have pauses between commands because in the development of high level software timing is rarely a concern. With robotics, as is the case in this project, timing of the software and hardware is important for the code to correctly control the hardware. Placing a pause after each command that uses the I2C bus addressed the questionable problems listed in [Table 3.16](#).

Code Usage

This chapter will explain in detail the methods developed for the gRAPI software. After all the functions that control a component have been explained a small sample program will show the correct usage of the functions.

Servo

Table 4.1
Servo class methods

| | Parameters | Description |
|--|------------|--|
| Servo(int handle, int low, int high, int pwm) | handle | returned value from open |
| | low | set the servos minimum value for operation |
| | high | set the servos maximum value for operation |
| | pwm | pin location on the robostix's PWM pins |
| void center () | | centers the wheels |
| void right (int percent) | percent | the percentage to the maximum value to the right |
| void left (int percent) | percent | the percentage to the maximum value to the left |
| int increase(int percent) | percent | amount percent is increased: percent += increase |
| int decrease(int percent) | percent | amount percent is decreased: percent -= decrease |

[Table 4.1 lists all the methods for the Servo class and the parameters passed for each method with a brief description of each parameter or the method. Each method is explained in more detail below.](#)

Servo(int handle, int low, int high, int pwm) - the constructor for the Servo class.

There are four parameters that are passed into the constructor: handle, low,

high, and pwm. The handle is the integer value returned by the open command for the I2C bus. The low and high values are to set the pulse width (PWM) signal range for the servo. This allows the code to control servos with different pulse width signal ranges. The pwm parameter corresponds to the pin location on the robostix™'s PWM pins where the servo is connected. This command allows a user of the code to setup a servo for use in a program.

center() - method used for centering the wheels during operation of the robot.

Method moves the servo to its center position.

right(percent) - method used to turn the robot to the right. The percent value passed to this method is how far the wheels will be turned to the right.

The value of percent ranges from 0 - 100. A value of zero is the same as issuing the center() command, and a value of 100 will turn the wheels to the maximum right position.

left(percent) - method used to turn the robot to the left. The percent value passed

to this method is how far the wheels will be turned to the left. The value of percent ranges from 0 - 100. A value of zero is the same as issuing the center() command, and a value of 100 will turn the wheels to the maximum right position.

increase(percent) - The amount that current percent is increased for the servo. If

current percent is set to a value of 10 and the increase method was used and a value of 10 was passed into the increase method, percent would be increased to 20.

decrease(percent) - The amount that current percent is decreased for the servo.

If current percent is set to a value of 20 and the decrease method was used and a value of 10 was passed into the decrease method, percent would be decreased to 10.

The following is an example program showing how to use the methods in the servo class.

```
/*
  testServoRL.cpp

  Moves wheels to max right, then center, then max left.
  Then moves incrementally from max left to max right.
*/

#include <iostream>
using namespace std;

#include <stdlib.h>

#include "Servo.h"

const int maxLeft = 100;
const int maxRight = 100;
const int inc = 1;

int main()
{
  int i2cHandle;
  int i;
```

```

// opening the I2C bus for reading and writing
// and storing its handle value
if((i2cHandle = open("/dev/i2c-0", O_RDWR)) < 0)
{
    cout << "Open Failed" << endl;
    exit(1);
}

Servo turn(i2cHandle, 1100, 1894, 3); // centers wheels

turn.right(maxRight);
sleep(2);

turn.center();
sleep(2);

turn.left(maxLeft);
sleep(2);

// move wheels from max left to center
for(i = maxLeft; i > 0; i-=inc)
{
    turn.left(i);
    usleep(1000);
}

// move wheels from center to max right
for(i = 0; i <= maxRight; i+=inc)
{
    turn.right(i);
    usleep(1000);
}
sleep(3);
turn.center();

return 0;
}

```

Motor

Table 4.2
Motor class methods

| | Parameters | Description |
|---|------------|---|
| Motor(int handle, int low, int high, int pwm, int delay) | handle | returned value from open |
| | low | set the motors minimum value for operation |
| | high | set the motors maximum value for operation |
| | pwm | pin location on the robostix's PWM pins |
| | delay | how often a pulse is sent to the motor |
| void stop() | | stops the movement of the motor |
| void forward(int percent) | percent | the percentage to the maximum value for forward |

| | Parameters | Description |
|------------------------------------|------------|--|
| void reverse(int percent) | percent | the percentage to the maximum value for reverse |
| int increase(int percent) | percent | amount percent is increased: percent += increase |
| int decrease(int percent) | percent | amount percent is decreased: percent -= decrease |

[Table 4.2 lists all the methods for the Motor class and the parameters passed for each method with a brief description of each parameter or the method. Each method is explained in more detail below.](#)

Motor(int handle, int low, int high, int pwm, int delay) - the constructor for the Motor class. There are five parameters that are passed into the constructor: handle, low, high, pwm, and delay. The handle is the integer value returned by the open command for the I2C bus. The low and high values are to set the pulse width (PWM) signal range for the motor. This allows the code to control servos with different pulse width signal ranges. The pwm parameter corresponds to the pin location on the robostix™'s PWM pins where the motor is connected. The delay parameter corresponds to how often the motor is pulsed. If a delay of 2 is given the motor will be pulsed every other time. This command allows a user of the code to setup a motor for use in a program.

void stop() - method used to stop the motor from going forward or in reverse.

This function does not ensure that the car will completely stop moving

when the command is issued, it only puts the motor controller into the stop position.

`void forward(percent)` - method used to move the robot to the forward direction.

The percent value passed to this method is how fast the robot will move.

The value of percent ranges from 0 - 100. A value of zero is the same as issuing the `stop()` command, and a value of 100 will be full speed for the robot.

`void reverse(percent)` - method used to move the robot to the reverse direction.

The percent value passed to this method is how fast the robot will move.

The value of percent ranges from 0 - 100. A value of zero is the same as issuing the `stop()` command, and a value of 100 will be full speed for the robot.

`int increase(percent)` - The amount that current percent is increased for the motor.

If current percent is set to a value of 10 and the increase method was used and a value of 10 was passed into the increase method, percent would be increased to 20.

`int decrease(percent)` - The amount that current percent is decreased for the

motor. If current percent is set to a value of 20 and the decrease method

was used and a value of 10 was passed into the decrease method, percent would be decreased to 10.

The following is an example program showing how to use the methods in the motor class.

```
/*
  testSpeedFwRv.cpp

  moves wheels minForwd
  then moves wheels minReverse
*/

#include <iostream>
using namespace std;

#include <stdlib.h>

#include "Motor.h"
#include "Servo.h"

const int minForward = 12;
const int minReverse = 20;

int main()
{
  int i2cHandle;

  // opening the I2C bus for reading and writing
  // and storing its handle value
  if((i2cHandle = open("/dev/i2c-0", O_RDWR)) < 0)
  {
    cout << "Open Failed" << endl;
    exit(1);
  }

  Motor go(i2cHandle, 1100, 1900, 4, 3);
  Servo turn(i2cHandle, 1100, 1900, 3); // centers wheels

  go.forward(minForward);
  sleep(3);

  go.stop();
  sleep(1);

  go.reverse(minReverse);
  sleep(3);

  go.stop();
}
```

```

    cout << "Test complete" << endl;
    return 0;
}

```

SRF10

Table 4.3
Srf10 class methods

| | Parameters | Description |
|---|------------|--|
| Srf10(int handle, int addr, char resultType) | handle | returned value from open |
| | addr | device address used on the I2C bus |
| | resultType | stored buffer value type: c = centimeter, i = inches, s = microsecond |
| void ping() | | starts the ranging and clears the buffer |
| int readS() | | get the value from the buffer |
| void setGain(int gainValue) | gainValue | new gain value for sensor: from 0 - 16; 0 = gain of 40, 16 = gain of 7 |
| void setRange(int rangeValue) | rangeValue | new rage value for sensor: ((Range Register x 43mm) + 43mm) |
| void setResultType(char resultType) | resultType | stored buffer value type: c = centimeter, i = inches, s = microsecond |
| void changeAddress(int newAddress) | newAddress | new address to be assigned to sensor: Hex; E0, E2, E4, ..., FE |

[Table 4.3 lists all the methods for the Srf10 class and the parameters passed for each method with a brief description of each parameter or the method. Each method is explained in more detail below.](#)

Srf10(int handle, int addr, char resultType) - the constructor for the Srf10 class.

There are three parameters that are passed into the constructor: handle, addr, and resultType. The handle is the integer value returned by the open command for the I2C bus. The addr parameter is the SRF10 sensor address that has been assigned when the sensor was setup to work with multiple sensors on an I2C bus. The resultType parameter specifies how the range value will be stored: inches, centimeters, or microseconds.

void ping() - the method used to initialize the sensor, clear the buffer, and start the ranging of the sensor.

int readS() - the method used to get the value out of the buffer for use in a program.

void setGain(gainValue) - the method used to change the gain value of the sensor from the default value.

void setRange(rangeValue) - the method used to change the range of the sensor from the default value.

void setResultType(resultType) - the method to change the value type of the value stored in the buffer.

`void changeAddress(newAddress)` - method used to change the address of a sensor. This will only work when one sensor is connected to the I2C bus. Can not change the address of a sensor when multiple sensors are on the I2C bus.

The following is an example program showing how to use the methods in the `srf10` class.

```
/*
   testSensorRead.cpp

   reads sensor and displays value in inches, centimeters, or
   microseconds
*/

#include <iostream>
using namespace std;

#include <stdlib.h>

#include "Srf10.h"

int main()
{
    int i2cHandle;

    // opening the I2C bus for reading and writing
    // and storing its handle value
    if((i2cHandle = open("/dev/i2c-0", O_RDWR)) < 0)
    {
        cout << "Open Failed" << endl;
        exit(1);
    }

    /*
       There are 16 addresses available to the srf10 range sensor
       (0-15). The changeAddress method will take a number from
       0 - 15 and perform the necessary commands to change the
       address of that sensor.
    */
}
```

```

int i2cAddr;
char resultType; // i = inches, c = centimeters,
                 // s = microseconds

cout << "Enter sensor address (0 - 15): ";
cin >> i2cAddr;
i2cAddr += 0x70;

cout << "Enter result type (i, c, s): ";
cin >> resultType;

if(i2cAddr >= 0x70 && i2cAddr <= 0x7F)// checking for valid
                                       // I2C address

    switch(resultType)
    {
        case 'i':
        case 'I':
        case 'c':
        case 'C':
        case 's':
        case 'S':
            break;
        default:
            cout << "Invalid result type" << endl;
            exit(1);
    }
else
{
    cout << "Invalid i2c address (0 - 15)" << endl;
    exit(1);
}

Srf10 sensor(i2cHandle, i2cAddr, resultType);

// changing the gain and range of sensor from its default
sensor.setGain(9);
usleep(1000);
sensor.setRange(47);
usleep(1000);

while(1)
{
    sensor.ping();
    usleep(65000);

    cout << "Distance: " << sensor.readS();
    switch(resultType)
    {
        case 'i':
        case 'I':
            cout << " inches" << endl;
            break;
        case 'c':
        case 'C':
            cout << " centimeters" << endl;
            break;
        case 's':
        case 'S':
    }
}

```

```

        cout << " microseconds" << endl;
        break;
    }
    usleep(1000);
}
return 0;
}

```

Compass

Table 4.4
Cmps03 class methods

| | Parameters | Description |
|-----------------------------|------------|-------------------------------|
| Cmps03(int handle) | handle | returned value from open |
| double readC() | | get the value from the buffer |
| int calibrate() | | calibrate the sensor |

[Table 4.4 lists all the methods for the Cmps03 class and the parameters passed for each method with a brief description of each parameter or the method. Each method is explained in more detail below.](#)

Cmps03(int handle) - the constructor for the Cmps03 class. There is one parameters that is passed into the constructor: handle. The handle is the integer value returned by the open command for the I2C bus.

`double readC()` - the method used to get the value out of the buffer for use in a program. Value returned is in degrees between 0 and 359.9.

`int calibrate()` - the method used to calibrate the compass sensor to North, East, South, and West. The method prompts the user to position the robot in the specified direction.

The following is an example program showing how to use the methods in the `Cmps03` class.

```
/*
   testCompass.cpp

   Read and display value from the compass in infinite loop
*/

#include <iostream>
using namespace std;

#include <stdlib.h>

#include "Cmps03.h"

int main()
{
    int i2cHandle;

    // opening the I2C bus for reading and writing
    // and storing its handle value
    if((i2cHandle = open("/dev/i2c-0", O_RDWR)) < 0)
    {
        cout << "Open Failed" << endl;
        exit(1);
    }

    Cmps03 compass(i2cHandle);

    float deg;
```

```
while(1)
{
    deg = compass.readC();
    usleep(1000);
    printf("Deg: %.1f\n", deg);
}

return 0;
}
```

The full gRAPI source code is in appendix B.

Conclusion

The goal of this project was to develop gRAPI, an application programming interface (API) for a gumstix connex™ motherboard in order to control an indoor four-wheeled robot. With the development of gRAPI, ISL will have the needed control software for their Hallway Hummer project. ISL can continue research and development on the other goals for the Hallway Hummer using the gumstix connex™.

Specific hardware was selected for this project: robot platform, sensors, and controller. gRAPI was developed to control the selected hardware. Testing of the software was done in two phases. The first phase was to test each component individually for proper functionality and control. Phase two of testing was to test multiple components together in a single program. gRAPI successfully passed all tests for each phase.

The Hallway Hummer project that ISL is currently developing needed a more powerful controller. The gumstix connex™ motherboard was selected for the Hallway Hummer project and is in the integration phase with that controller. With the development of gRAPI for a gumstix connex™ motherboard, more advanced robotics software for mapping and localization, vision, and other algorithms can be implemented. The ISL Hallway Hummer project will be able to use gRAPI to pursue other research goals for the use of autonomous robots in search and rescue.

REFERENCE LIST

- [1] IRIS. "Welcome." Retrieved February 21, 2009 from the World Wide Web: <http://iris.ecst.csuchico.edu/welcome/index.htm>.
- [2] Robin R. Murphy, *Introduction to AI Robotics*. Cambridge, Massachusetts: The MIT Press, 2000.
- [3] Gumstix. "gumstix connex 400xm-bt (with duck antenna)." Retrieved February 14, 2009 from the World Wide Web: http://gumstix.com/store/catalog/product_info.php?products_id=137.
- [4] ISL. "The OCNL Hallway Gumstix/Hummer." Retrieved February 14, 2009 from the World Wide Web: <http://isl.ecst.csuchico.edu/projects/hummer01.htm>.
- [5] ISL. "The IRIS Minimoto electric ATV SAR Robot." Retrieved February 14, 2009 from the World Wide Web: <http://isl.ecst.csuchico.edu/projects/minimoto01.htm>.
- [6] P.J. McKerrow, *Introduction to Robotics*. Addison Wesley, Sydney, 1991.
- [7] Webopedia. "API." Retrieved February 14, 2009 from the World Wide Web: <http://www.webopedia.com/TERM/A/API.html>.
- [8] Begum, Momotaz, Mann, George K.I., and Gosine, Raymond G. "Integrated fuzzy logic and genetic algorithmic approach for simultaneous localization and mapping of mobile robots," *Applied Soft Computing*, 8, pp. 150-165, 2008.
- [9] Ho, Kin Leong, and Newman, Paul. "Loop closure detection in SLAM by combining visual and spatial appearance," *Robotics and Autonomous Systems*, 54, pp. 740-749, 2006.
- [10] Castellanos, J.A., Martinez-Cantin, R., Tardós, J.D., and Neira, J. "Robocentric map joining: Improving the consistency of EKF-SLAM," *Robotics and Autonomous Systems*, 55, pp. 21-29, 2007.
- [11] Rodriguez-Losada, Diego, Matia, Fernando, and Galan, Ramon. "Building geometric feature based maps for indoor service robots," *Robotics and Autonomous Systems*, 54, pp. 546-558, 2006.

- [12] Grisetti, Giorgio, Tipaldi, Gian Diego, Stachniss, Cyrill, Burgard, Wolfram, and Nardi, Daniele. "Fast and accurate SLAM with Rao--Blackwellized particle filters," *Robotics and Autonomous Systems*, 55, pp. 30-38, 2007.
- [13] Milford, Michael, Schulz, Ruth, Prasser, David, Wyeth, Gordon, and Wiles, Janet. "Learning spatial concepts from RatSLAM representations," *Robotics and Autonomous Systems*, 55, pp. 403-410, 2007.
- [14] Austin, David J., and McCarragher, Brennan J. "Geometric constraint identification and mapping for mobile robots," *Robotics and Autonomous Systems*, 35, pp. 59-76, 2001.
- [15] Remazeilles, Anthony, and Chaumette, François. "Image-based robot navigation from an image memory," *Robotics and Autonomous Systems*, 55, pp. 345-356, 2007.
- [16] Vasudevan, Shrihari, Gächter, Stefan, Nguyen, Viet, Siegwart, Roland. "Cognitive maps for mobile robots -- an object based approach," *Robotics and Autonomous Systems*, 55, pp. 359-371, 2007.
- [17] Wang, Meng, Liu, James N.K. "Fuzzy logic-based real-time robot navigation in unknown environment with dead ends," *Robotics and Autonomous Systems*, In Press, Corrected Proof, Oct. 2007.
- [18] J. L. Jones and A. M. Flynn, *Mobile Robots: Inspiration to Implementation*. A. K. Peters Ltd, Wellesley, Mass., 1993.
- [19] Brooks, R. A. "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, RA-2(1):14-23, 1986.
- [20] Sowmya, Arcot, So, David Tsz-Wang, Tang, Wan Hung. "Design of a Mobile Robot Controller using Esterel Tools," *Theoretical Computer Science*, 65(5), pp. 3-10, 2002.
- [21] Hong, Keum-Shik, Choi, Kyung-Hyun, Kim, Jeom-Goo, Lee, Suk. "A PC-based open robot control system: PC-ORC," *Robotics and Computer Integrated Manufacturing*, 17, pp. 355-365, 2001.
- [22] Traxxas™. "Traxxas E-Maxx." Retrieved February 14, 2009 from the World Wide Web:
http://www.traxxas.com/products/electric/emaxx/trx_emaxx_specs.htm.

- [23] Gumstix. "tweener." Retrieved February 14, 2009 from the World Wide Web: http://gumstix.com/store/catalog/product_info.php?products_id=106.
- [24] Gumstix. "robostix." Retrieved February 14, 2009 from the World Wide Web: http://gumstix.com/store/catalog/product_info.php?cPath=31&products_id=139.
- [25] Gumstix. "wifistix FCC (with wifi antenna)." Retrieved February 14, 2009 from the World Wide Web: http://gumstix.com/store/catalog/product_info.php?cPath=31&products_id=171.
- [26] Acroname Robotics. "Devantech SRF10 Ranger." Retrieved February 14, 2009 from the World Wide Web: <http://www.acroname.com/robotics/parts/R241-SRF10.html>.
- [27] Acroname Robotics. "Devantech Compass." Retrieved February 14, 2009 from the World Wide Web: <http://www.acroname.com/robotics/parts/R117-COMPASS.html>.
- [28] Gumstix. "Buildroot." Retrieved February 14, 2009 from the World Wide Web: <http://docwiki.gumstix.org/index.php/Buildroot>.
- [29] Ubuntu. "Ubuntu." Retrieved February 14, 2009 from the World Wide Web: <http://www.ubuntu.com/>.
- [30] Traxxas™. "Accessories: Traxxas Electronic Servos." Retrieved February 19, 2009 from the World Wide Web: http://www.traxxas.com/products/accessories/trx_accessories_servos.htm.
- [31] Traxxas™. "Accessories: EVX Electronic FWD/REV speed control." Retrieved February 19, 2009 from the World Wide Web: http://www.traxxas.com/products/accessories/trx_accessories_evx.htm.
- [32] Gumstix. "I2C on the Gumstix." Retrieved February 14, 2009 from the World Wide Web: <http://docwiki.gumstix.org/index.php/I2c>.
- [33] Gumstix. "Robostix." Retrieved February 14, 2009 from the World Wide Web: <http://docwiki.gumstix.org/index.php/Robostix>.
- [34] Gumstix. "Robostix simple servo." Retrieved February 14, 2009 from the World Wide Web: http://docwiki.gumstix.org/index.php/Robostix_simple_servo.

[35] Gumstix. "Robostix samples." Retrieved February 14, 2009 from the World Wide Web: http://docwiki.gumstix.org/index.php/Robostix_samples.

APPENDIX A

Preparation of ISL's Hallway Hummer to use gRAPI

Hardware Analysis

- Test servo to find the minimum and maximum of the range of operation
(Example: min.: 1ms, max.: 2ms)
- Test motor controller to find the minimum and maximum of the range of operation
(Example: min.: 1ms, max.: 2ms)

The following steps for computer setup and the gumstix connex™ and robostix™ setup should be followed from the detailed instructions provided by Gumstix at their Wiki pages at this address: <http://docwiki.gumstix.org/index.php/Index.html>.

Computer Setup

- Here is the first page explaining computer setup: <http://docwiki.gumstix.org/index.php/Buildroot>
- I followed the Ubuntu instructions because that is what I was using. Follow the instructions for the correct OS that you are using. Here is the link to the Ubuntu setup: http://docwiki.gumstix.org/index.php/Buildroot_on_Ubuntu

gumstix connex and robostix setup

gumstix connex™

U-Boot information is found here:

http://docwiki.gumstix.org/index.php/U-Boot#Booting_the_kernel

Information on how to transfer files:

<http://docwiki.gumstix.org/index.php/Tutorial>

The gumstix connex™ setup is information on how to set the U-Boot for loading a new kernel to the gumstix connex™ motherboard. The next link is information on how to transfer programs that have been written and need to be tested to the gumstix connex™.

robostix™

Information on the robostix™:

<http://docwiki.gumstix.org/index.php/Robostix>

Information on tweener™ use with robostix™ and gumstix connex™:

http://docwiki.gumstix.org/index.php/Tweener_modifications

To program the robostix™ an AVR programmer is needed. I purchased the AVR Studio kit to do the programming of the robostix™ for the gRAPI project. After programming the robostix™ I used the i2c bootloader program that is found here http://docwiki.gumstix.org/index.php/Robostix_i2c_bootloader to do any program changes to the robostix™ after the initial programming.

With all the above steps done and or understood, one can now load the software used for gRAPI on the robostix™ and development of code can be started. Here is a link to the software developed for gRAPI that is saved in my area for ISL: <http://isl.ecst.csuchico.edu/DOCS/Logs/Michael/MastersProject/>. In that folder there is another folder named robostix. In the robostix folder is located all the code that is needed for the robostix and gumstix. Software for the robostix is located in the robostix folder. There should be a compiled .hex file that can be transferred to the robostix by one of the methods mentioned above. You need to use this file or recompile the file and only this file should be used on the robostix. Because, modifications have been made so that gRAPI works with that file that resides on the robostix. There is a gumstix folder in the robostix folder which contains programs that run on the gumstix. Software for gRAPI is located in the project folder inside of the gumstix folder. All the source code and sample programs are located in this folder that were written for the gRAPI project. There is a Makefile that is also located in the project folder and was used to compile all the test programs by changing the file name variable located at the top of the Makefile to the correct program that needed to be compiled. If development of software with gRAPI is desired in a different location than specified, modifications to the Makefile will be needed. Make sure you know how to make the necessary changes to cross-compile the code and use gRAPI. The scp program was used to copy the executable to the gumstix connex™ after the program was compiled. The steps for compiling and transferring the executable are repeated as necessary.

With the following completed everything should be working and development with gRAPI can take place.

APPENDIX B

Source Code for gRAPI

```
/*
Michael Simmons
Master Project:
  gRAPI: an indoor four-wheeled robot API for a gumstix connex

Last Modified: 17 Feb 2009

Header file for the Cmps03 class
*/

#ifndef CMPS03_H
#define CMPS03_H

#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/i2c.h>
#include <unistd.h>
#include <inttypes.h>

class Cmps03
{
private:
  int i2cHandle; // file handle from open of I2C
  int i2cAddr; // address on I2C bus
  uint8_t readBuf[2]; // result value
  uint8_t readReg; // value of 2 is register to read from
  int i2cBusWait;

public:
  Cmps03(int handle);
  double readC(); // read stored register value
  int calibrate(); // sets direction for North, East, South, West
}; // Cmps03

#endif
```

```

/*
Michael Simmons
Master Project:
  gRAPI: an indoor four-wheeled robot API for a gumstix connex

Last Modified: 17 Feb 2009

Cmps03 class
*/

#include "Cmps03.h"
#include <iostream>

using namespace std;

// Constructor
Cmps03::Cmps03(int handle)
{
  readReg = 2;
  i2cHandle = handle;
  i2cAddr = 0x60;
  i2cBusWait = 10;
} // Cmps03

double Cmps03::readC()
{
  float value;

  ioctl(i2cHandle, I2C_SLAVE, i2cAddr);

  write(i2cHandle, &readReg, 1);
  usleep(i2cBusWait);
  read(i2cHandle, readBuf, 2);
  usleep(i2cBusWait);
  value = readBuf[0]*256 + readBuf[1];

  value = value/10.0;

  return value;
} // readC

int Cmps03::calibrate()
{
  int i,
      direction;
  uint8_t calReg = 15;
  uint8_t writeBuf[2];

  writeBuf[0] = calReg;
  writeBuf[1] = 255;

  for(i = 4; i > 0; i--)
  {
    do
    {
      switch(i)
      {
        case 4: cout << "Point vehicle North and enter a 4" << endl;

```

```
        break;
    case 3: cout << "Point vehicle East and enter a 3" << endl;
            break;
    case 2: cout << "Point vehicle South and enter a 2" << endl;
            break;
    case 1: cout << "Point vehicle West and enter a 1" << endl;
            break;
    }
    cin >> direction;
} while (i != direction);

ioctl(i2cHandle, I2C_SLAVE, i2cAddr);
usleep(i2cBusWait);

write(i2cHandle, &writeBuf, 2);
usleep(i2cBusWait);
}
} // calibrate
```

```

/*
  Michael Simmons
  Master Project:
    gRAPI: an indoor four-wheeled robot API for a gumstix connex

  Last Modified: 17 Feb 2009

  Header file for the I2CDefs
*/

#ifndef I2CDEF_H
#define I2CDEF_H

#include <fcntl.h>
#include <sys/ioctl.h>
#include <stdint.h>
#include <linux/i2c.h>
#include <iostream>

using namespace std;

const uint8_t I2c_packet_len = 32;

struct I2c_packet_header_t
{
  uint16_t i2c_addr;
  unsigned short flags;
  short len; // number of bytes in packet
  char *packet;
}; // I2c_packet_header_t

struct I2c_pwm_packet_t
{
  uint8_t pwm_no; // 0=1a, 1=1b, 2=1c, 3=3a, 4=3b, 5=3c
  uint8_t pulseDelay;
  uint16_t pulse_value; // given in usec
}; // I2c_pwm_packet_t

struct I2c_ioctl_packet_t
{
  struct I2c_packet_header_t *addr;
  int packet_count;
}; // I2c_ioctl_packet_t

#endif

```

```

/*
Michael Simmons
Master Project:
  gRAPI: an indoor four-wheeled robot API for a gumstix connex

Last Modified: 17 Feb 2009

Header file for the I2cPWM class
*/

#ifndef I2CPWM_H
#define I2CPWM_H

#include "I2CDefs.h"

class I2cPWM
{
private:
  // all pulse values in usec
  uint16_t bottomValue;
  uint16_t midValue;
  uint16_t topValue;
  uint16_t rangeValue;
  uint16_t newValue;
  uint8_t pulseDelay;

  int i2cHandle; //file handle from open of I2C
  int avrAddr; //address of avr chip on I2C

  int i2cBusWait;

  uint8_t pwmNum; //pwm pin location on robostix

  I2c_ioctl_packet_t packet;
  I2c_packet_header_t packet_header;
  I2c_pwm_packet_t packet_data;

  uint8_t buf[I2c_packet_len];

  void sendPacket();

public:
  I2cPWM(int handle, int low, int high, int pwm, int delay);
  void middle();
  void increase(int percent);
  void decrease(int percent);
}; // I2cPwm

#endif // I2CPWM_H

```

```

/*
  Michael Simmons
  Master Project:
    gRAPI: an indoor four-wheeled robot API for a gumstix connex

  Last Modified: 17 Feb 2009

  I2cPWM class
*/

#include "I2cPWM.h"

// Constructor
I2cPWM::I2cPWM(int handle, int low, int high, int pwm, int delay)
{
  i2cHandle = handle;
  bottomValue = low;
  topValue = high;
  pwmNum = pwm;
  pulseDelay = delay;
  avrAddr = 0x0b; // address of avr chip on the I2C bus
  i2cBusWait = 10;

  midValue = (low + high) / 2;
  rangeValue = (high - low) / 2;

  //setup packet header
  packet_header.i2c_addr = avrAddr;
  packet_header.flags = 0;
  packet_header.len = sizeof(packet_data) + 2;
  packet_header.packet = (char *)&buf[0];

  packet.addr = &packet_header;
  packet.packet_count = 1;

  packet_data.pwm_no = pwmNum;
  packet_data.pulseDelay = pulseDelay;

  middle();
} // I2cPwm

void I2cPWM::sendPacket()
{
  if(ioctl(i2cHandle, I2C_SLAVE, avrAddr))
    cout << "Error slaving avr" << endl;
  usleep(i2cBusWait);

  buf[0] = 0;
  buf[1] = sizeof(packet_data);
  memcpy(&buf[2], (void *)&packet_data, sizeof(packet_data));

  if (ioctl(i2cHandle, I2C_RDWR, &packet) < 0)
  {
    cout << "Error writing to avr" << endl;
  }
  usleep(i2cBusWait);
} // sendPacket

```

```
void I2cPWM::middle()
{
    packet_data.pulse_value = midValue;

    sendPacket();
} // middle

void I2cPWM::increase(int percent)
{
    if(percent > 100)
        percent = 100;
    else
        if(percent < 0)
            percent = 0;

    newValue = midValue + (percent * rangeValue / 100);

    packet_data.pulse_value = newValue;

    sendPacket();
} // increase

void I2cPWM::decrease(int percent)
{
    if(percent > 100)
        percent = 100;
    else
        if(percent < 0)
            percent = 0;

    newValue = midValue - (percent * rangeValue / 100);

    packet_data.pulse_value = newValue;

    sendPacket();
} // decrease
```

```
/*
  Michael Simmons
  Master Project:
    gRAPI: an indoor four-wheeled robot API for a gumstix connex

  Last Modified: 17 Feb 2009

  Header file for the Motor class
*/

#ifndef MOTOR_H
#define MOTOR_H

#include "I2cPWM.h"

class Motor : public I2cPWM
{
  private:
    int currentPer;
    enum {backwd, stopped, forwd}direction;

  public:
    Motor(int handle, int low, int high, int pwm, int delay);
    void stop();
    void forward(int percent);
    void reverse(int percent);
    int increase(int percent);
    int decrease(int percent);
}; // Motor

#endif
```

```

/*
  Michael Simmons
  Master Project:
    gRAPI: an indoor four-wheeled robot API for a gumstix connex

  Last Modified: 17 Feb 2009

  Motor class
*/

#include "Motor.h"

// Constructor
Motor::Motor(int handle, int low, int high, int pwm, int delay) :
    I2cPWM(handle, low, high, pwm, delay)
{
    currentPer = 0;
    direction = stopped;
} // Motor

void Motor::stop()
{
    currentPer = 0;
    direction = stopped;
    I2cPWM::middle();
} // stop

void Motor::forward(int percent)
{
    I2cPWM::increase(percent);
    currentPer = percent;
    direction = forwd;
} // forward

void Motor::reverse(int percent)
{
    I2cPWM::decrease(percent);
    currentPer = percent;
    direction = backwd;
} // reverse

int Motor::increase(int percent)
{
    int flag;

    currentPer += percent;
    if(currentPer > 100)
    {
        currentPer = 100;
        flag = -1;
    }
    else flag = 0;

    switch(direction)
    {
        case forwd:
            forward(currentPer);
            break;
    }
}

```

```

        case stopped:
            forward(currentPer);
            break;
        case backwd:
            reverse(currentPer);
            break;
    }
    return flag;
} // increase

int Motor::decrease(int percent)
{
    currentPer -= percent;
    if(currentPer <= 0)
    {
        currentPer = 0;
        direction = stopped;
        stop();
        return -1;
    }

    switch(direction)
    {
        case forwd:
            forward(currentPer);
            break;
        case stopped:
            reverse(currentPer);
            break;
        case backwd:
            reverse(currentPer);
            break;
    }
    return 0;
} // decrease

```

```
/*
  Michael Simmons
  Master Project:
    gRAPI: an indoor four-wheeled robot API for a gumstix connex

  Last Modified: 17 Feb 2009

  Header file for the Servo class
*/

#ifndef SERVO_H
#define SERVO_H

#include "I2cPWM.h"

class Servo : public I2cPWM
{
  private:
    int currentPer;
    enum {lef, cent, rite}direction;

  public:
    Servo(int handle, int low, int high, int pwm);
    void center();
    void right(int percent);
    void left(int percent);
    int increase(int percent);
    int decrease(int percent);
}; // Servo

#endif
```

```

/*
  Michael Simmons
  Master Project:
    gRAPI: an indoor four-wheeled robot API for a gumstix connex

  Last Modified: 17 Feb 2009

  Servo class
*/

#include "Servo.h"

// Constructor
Servo::Servo(int handle, int low, int high, int pwm) :
    I2cPWM(handle, low, high, pwm, 1)
{
    currentPer = 0;
    direction = cent;
} // I2cPwm

void Servo::center()
{
    currentPer = 0;
    direction = cent;
    I2cPWM::middle();
} // center

void Servo::right(int percent)
{
    I2cPWM::increase(percent);
    currentPer = percent;
    direction = rite;
} // right

void Servo::left(int percent)
{
    I2cPWM::decrease(percent);
    currentPer = percent;
    direction = lef;
} // left

int Servo::increase(int percent)
{
    int flag;

    switch(direction)
    {
        {
            currentPer += percent;
            if(currentPer > 100)
            {
                currentPer = 100;
                flag = -1;
            }
            else flag = 0;

            case rite:
                right(currentPer);
                break;

```

```

        case cent:
            right(currentPer);
            break;
        case lef:
            left(currentPer);
            break;
    }
    return flag;
} // increase

int Servo::decrease(int percent)
{
    switch(direction)
    {
        {
            currentPer -= percent;
            if(currentPer <= 0)
            {
                currentPer = 0;
                direction = cent;
                center();
                return -1;
            }

            case rite:
                right(currentPer);
                break;
            case cent:
                left(currentPer);
                break;
            case lef:
                left(currentPer);
                break;
        }
    }
    return 0;
} // decrease

```

```

/*
Michael Simmons
Master Project:
  gRAPI: an indoor four-wheeled robot API for a gumstix connex

Last Modified: 17 Feb 2009

Header file for the Srf10 class
*/

#ifndef SRF10_H
#define SRF10_H

#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/i2c.h>
//#include <stdio.h>
//#include <stdlib.h>
#include <unistd.h>

class Srf10
{
private:
  int i2cHandle; // file handle from open of I2C
  int i2cAddr; // address on I2C bus
  char pingBuf[2]; // result specifier
  char readBuf[2]; // result value
  char gainBuf[2]; // set gain
  char rangeBuf[2]; // set range
  char readReg; // value of 2 is register to read from
  int i2cBusWait;

public:
  Srf10(int handle, int addr, char rt = 'c' );
  void ping(); // fire sensor
  int readS(); // read stored register value
  void setGain(int); // set sensitivity to signal,
    // 8 = gain 140, or 2 meters
  void setRange(int); // set range (Reg * 43 mm + 43 mm),
    // 24 = 1 meter, 47 = 2 meters,
    // 70 = 3 meters, 93 = 4 meters
  void setResultType(char); // i = inches, c = centimeters,
    // s = micro-seconds

/*
** NOTE **: only one sensor can be on the i2c bus when changing an
address

There are 16 addresses available to the srf10 range sensor (0-15).
The changeAddress method will take a number from 0 - 15 and perform
the necessary commands to change the address of that sensor.

** REMEMBER **: only one sensor can be on the i2c bus when changing
an address
*/

```



```

/*
Michael Simmons
Master Project:
  gRAPI: an indoor four-wheeled robot API for a gumstix connex

Last Modified: 17 Feb 2009

  Srf10 class
*/

#include "Srf10.h"

// Constructor
Srf10::Srf10(int handle, int addr, char rt )
{
  readReg = 2;
  pingBuf[0] = 0;
  gainBuf[0] = 1;
  rangeBuf[0] = 2;
  i2cHandle = handle;
  i2cAddr = addr;
  i2cBusWait = 10;

  setResultType(rt);
} // Srf10

void Srf10::ping()
{
  ioctl(i2cHandle, I2C_SLAVE, i2cAddr);
  usleep(i2cBusWait);

  write(i2cHandle, pingBuf, 2);
  usleep(i2cBusWait);
} // ping

int Srf10::readS()
{
  ioctl(i2cHandle, I2C_SLAVE, i2cAddr);
  usleep(i2cBusWait);

  write(i2cHandle, &readReg, 1);
  usleep(i2cBusWait);
  read(i2cHandle, readBuf, 2);
  usleep(i2cBusWait);

  return readBuf[0]*256 + readBuf[1];
} // readS

void Srf10::setGain(int g)
{
  ioctl(i2cHandle, I2C_SLAVE, i2cAddr);
  usleep(i2cBusWait);

  gainBuf[1] = g;
  write(i2cHandle, &gainBuf, 2);
  usleep(i2cBusWait);
} // setGain

```

```

void Srf10::setRange(int r)
{
    ioctl(i2cHandle, I2C_SLAVE, i2cAddr);
    usleep(i2cBusWait);

    rangeBuf[1] = r;
    write(i2cHandle, &rangeBuf, 2);
    usleep(i2cBusWait);
} // setRange

void Srf10::setResultType(char rt)
{
    switch(rt)
    {
        case 'i':
        case 'I':
            pingBuf[1] = 80;
            break;
        case 'c':
        case 'C':
            pingBuf[1] = 81;
            break;
        case 's':
        case 'S':
            pingBuf[1] = 82;
            break;
        default:
            pingBuf[1] = 81;
            break;
    }
} // setResultType

/*
There are 16 addresses available to the srf10 range sensor (0-15).
The changeAddress method will take a number from 0 - 15 and perform
the necessary commands to change the address of that sensor.
*/
void Srf10::changeAddress(int newAddress)
{
    //newAddress is the exact Hex address in the srf10 manual
    int bufVal1 = 0xA0;
    int bufVal2 = 0xAA;
    int bufVal3 = 0xA5;

    ioctl(i2cHandle, I2C_SLAVE, i2cAddr);
    usleep(i2cBusWait);

    pingBuf[1] = 0xA0;
    write(i2cHandle, &pingBuf, 2);
    usleep(i2cBusWait);

    pingBuf[1] = 0xAA;
    write(i2cHandle, &pingBuf, 2);
    usleep(i2cBusWait);

    pingBuf[1] = 0xA5;
    write(i2cHandle, &pingBuf, 2);
    usleep(i2cBusWait);
}

```

```
pingBuf[1] = (newAddress*2) + 0xE0;
write(i2cHandle, &pingBuf, 2);
usleep(i2cBusWait);
} // changeAddress
```

APPENDIX C

Phase two test programs

```

/*
  Michael Simmons
  Master Project:
    gRAPI: an indoor four-wheeled robot API for a gumstix connex

  Last Modified: 17 Feb 2009

  Figure 8 test program
*/

#include <iostream>

using namespace std;

#include <stdlib.h>

#include "Motor.h"
#include "Servo.h"
#include "Cmps03.h"

float deg;
int tolerance = 5; // north +- tolerance is good enough
int degChange = 10; // how many degrees change for each pass -
                    // forward or backward.
                    // Last pass will use tolerance
int minReverse = 20; // speed for backup movement
int minForward = 20; // speed for forward movement
int turnSetting = 75; // how much wheels are turned
int i2cBusWait = 1000;

void firstLoop(Cmps03 &compass, Servo &turn, Motor &go);
void secondLoop(Cmps03 &compass, Servo &turn, Motor &go);

int main()
{
  int i2cHandle;

  if((i2cHandle = open("/dev/i2c-0", O_RDWR)) < 0)
  {
    cout << "Open Failed" << endl;
    exit(1);
  }

  Motor go(i2cHandle, 1100, 1876, 4, 3);
  usleep(i2cBusWait);
  Servo turn(i2cHandle, 1100, 1894, 3);
  usleep(i2cBusWait);
  Cmps03 compass(i2cHandle);
  usleep(i2cBusWait);

  deg = compass.readC();
  usleep(i2cBusWait);

  firstLoop(compass, turn, go);
  secondLoop(compass, turn, go);

  go.stop();
  usleep(i2cBusWait);
}

```

```

    turn.center();
    usleep(i2cBusWait);

    return 0;
} // main

void firstLoop(Cmps03 &compass, Servo &turn, Motor &go)
{
    int passed360 = 0;
    float curDeg;
    int done = 0;

    turn.right(turnSetting);
    usleep(i2cBusWait);
    go.forward(minForward);
    usleep(i2cBusWait);

    while(!done)
    {
        curDeg = compass.readC();
        usleep(i2cBusWait);
        if(deg > 340)
        {
            if(curDeg < deg)
            {
                done = 1;
                passed360 = 1;
            }
        }
        else
            if(curDeg > deg + 10)
                done = 1;
    }

    done = 0;

    while(!done)
    {
        curDeg = compass.readC();
        usleep(i2cBusWait);
        if(!passed360)
        {
            if(deg > curDeg)
                passed360 = 1;
        }
        else
        {
            if(curDeg >= deg - 1)
                done = 1;
        }
    }
} // firstLoop

void secondLoop(Cmps03 &compass, Servo &turn, Motor &go)
{
    int passed0 = 0;
    float curDeg;
    int done = 0;

```

```
turn.left(turnSetting);
usleep(i2cBusWait);
go.forward(minForward);
usleep(i2cBusWait);

while(!done)
{
    curDeg = compass.readC();
    usleep(i2cBusWait);
    if(deg < 20)
    {
        if(curDeg > deg)
        {
            done = 1;
            passed0 = 1;
        }
    }
    else
        if(curDeg < deg - 10)
            done = 1;
}

done = 0;

while(!done)
{
    curDeg = compass.readC();
    usleep(i2cBusWait);
    if(!passed0)
    {
        if(deg < curDeg)
            passed0 = 1;
    }
    else
    {
        if(curDeg <= deg + 1)
            done = 1;
    }
}
} // secondLoop
```

```

/*
  Michael Simmons
  Master Project:
    gRAPI: an indoor four-wheeled robot API for a gumstix connex

  Last Modified: 17 Feb 2009

  Point North test program
*/

#include <iostream>
using namespace std;

#include <stdlib.h>

#include "Servo.h"
#include "Motor.h"
#include "Cmps03.h"

enum {backwd, stopped, forwd} direction;

int tolerance = 3; // north +- tolerance is good enough
int degChange = 5; // how many degrees change for each pass -
                    // forward or backward.
                    // Last pass will use tolerance
int minReverse = 21; // speed for backup movement
int minForward = 18; // speed for forward movement
int turnSetting = 50; // how much wheels are turned
int i2cBusWait = 1000; // how long to wait after an i2c command
int moveDist = 500;

void turnLeft(Cmps03 &compass, Servo &turn, Motor &go);
void turnRight(Cmps03 &compass, Servo &turn, Motor &go);

int main()
{
  int i2cHandle;

  if((i2cHandle = open("/dev/i2c-0", O_RDWR)) < 0)
  {
    cout << "Open Failed" << endl;
    exit(1);
  }

  float deg;

  Cmps03 compass(i2cHandle);

  Motor go(i2cHandle, 1100, 1876, 4, 3);
  usleep(i2cBusWait);
  Servo turn(i2cHandle, 1100, 1894, 3); // steering servo
  usleep(i2cBusWait);

  direction = backwd;

  deg = compass.readC();
  usleep(i2cBusWait);

```

```

while(!(deg >=0 && deg <= tolerance || deg >= (360 - tolerance)
      && deg < 360))
{
  if(deg <= 180)
  {
    if(deg <= 15)
      if(deg <= 5)
        turnSetting = 25;
      else
        turnSetting = 35;
    if(direction == backwd)
      turnRight(compass, turn, go);
    else
      turnLeft(compass, turn, go);
  }
  else
  {
    if(deg >= 345)
      if(deg >= 355)
        turnSetting = 25;
      else
        turnSetting = 35;
    if(direction == backwd)
      turnLeft(compass, turn, go);
    else
      turnRight(compass, turn, go);
  }
  sleep(2);
  deg = compass.readC();
  usleep(i2cBusWait);
}

go.stop();
usleep(i2cBusWait);
turn.center();
usleep(i2cBusWait);
deg = compass.readC();
printf("Final Heading Deg: %.1f\n", deg);
return 0;
} // main

void turnRight(Cmps03 &compass, Servo &turn, Motor &go)
{
  float deg;
  int i;

  turn.right(turnSetting);
  usleep(i2cBusWait);
  switch(direction)
  {
    case backwd:
      deg = compass.readC();
      usleep(i2cBusWait);
      cout << "current heading ";
      printf("%.1f\n", deg);

      go.reverse(minReverse);
      usleep(i2cBusWait * moveDist * tolerance);
  }
}

```

```

        break;
    case forwd:
        deg = compass.readC();
        usleep(i2cBusWait);
        cout << "current heading ";
        printf("%.1f\n", deg);

        go.forward(minForward);
        usleep(i2cBusWait * moveDist * tolerance);
    }
    go.stop();
    usleep(i2cBusWait);
    cout << "right stopping" << endl;
    if(direction == backwd)
        direction = forwd;
    else
        direction = backwd;
} // turnRight

void turnLeft(Cmps03 &compass, Servo &turn, Motor &go)
{
    float deg;
    int i;

    turn.left(turnSetting);
    usleep(i2cBusWait);
    switch(direction)
    {
        case backwd:
            deg = compass.readC();
            usleep(i2cBusWait);
            cout << "current heading ";
            printf("%.1f\n", deg);

            go.reverse(minReverse);
            usleep(i2cBusWait * moveDist * tolerance);
            break;
        case forwd:
            deg = compass.readC();
            usleep(i2cBusWait);
            cout << "current heading ";
            printf("%.1f\n", deg);

            go.forward(minForward);
            usleep(i2cBusWait * moveDist * tolerance);
    }
    go.stop();
    usleep(i2cBusWait);
    cout << "left stopping" << endl;
    if(direction == backwd)
        direction = forwd;
    else
        direction = backwd;
} // turnLeft

```

```

/*
  Michael Simmons
  Master Project:
    gRAPI: an indoor four-wheeled robot API for a gumstix connex

  Last Modified: 17 Feb 2009

  Wall Hugger test program
*/

#include <iostream>

using namespace std;

#include "Motor.h"
#include "Servo.h"
#include "RSnsr.h"

int optDistWall = 60; // value is in cm, optimal distance from wall
int minDistWall = optDistWall * 3 / 4; // value is in cm, optimal - 1/4
int maxDistWall = optDistWall * 5 / 4; // value is in cm, optimal + 1/4
int stopDist = 25; // value is in cm = 10 inches
int extreme = 75;
int moveDist = 40;
int i2cBusWait = 1000;
int sensorWait = 30;

int minReverse = 18; // speed for backup movement
int minForward = 16; // speed for forward movement

void process(RSnsr &sensorC, RSnsr &sensorL, Servo &turn, Motor &go);

int main()
{
  int i2cHandle;

  if((i2cHandle = open("/dev/i2c-0", O_RDWR)) < 0)
  {
    cout << "Open Failed" << endl;
    exit(1);
  }

  Motor go(i2cHandle, 1100, 1876, 4, 3);
  usleep(i2cBusWait);
  Servo turn(i2cHandle, 1100, 1894, 3); // steering servo
  usleep(i2cBusWait);
  RSnsr sensorC(i2cHandle, 0x72, 'c'); // center sensor
  usleep(i2cBusWait);
  RSnsr sensorL(i2cHandle, 0x70, 'c'); // left sensor
  usleep(i2cBusWait);

  // sensorL.setGain(9);
  // usleep(i2cBusWait);
  // sensorL.setRange(47);
  // usleep(i2cBusWait);
  sensorC.setGain(9);
  usleep(i2cBusWait);
  sensorC.setRange(47);

```

```

    usleep(i2cBusWait);

    int i;
    int n;

    cout << "How many times to loop: ";
    cin >> n;

    go.forward(minForward);
    usleep(i2cBusWait);
    for(i = 0; i < n; i++)
    {
        process(sensorC, sensorL, turn, go);
    }

    turn.center();
    usleep(i2cBusWait);
    go.stop();
    usleep(i2cBusWait);

    return 0;
} // main

void process(RSnsr &sensorC, RSnsr &sensorL, Servo &turn, Motor &go)
{
    static int preDist = 0;
    static int extremeFlag = 0;

    int distC;
    int distL;

    sensorC.ping();
    usleep(10000);
    sensorL.ping();
    usleep(20000);
    distC = sensorC.mReadS();
    usleep(i2cBusWait);
    distL = sensorL.mReadS();
    usleep(i2cBusWait);

    if(distC > stopDist)
    {
        cout << "distL: " << distL << endl;
        if(distL < minDistWall)
        {
            cout << "< minDistWall ";
            if(distL > preDist)
            {
                cout << "center" << endl;
                turn.center();
                usleep(i2cBusWait);
            }
            else
            {
                cout << "extreme right" << endl;
                turn.right(extreme);
                usleep(i2cBusWait);
                extremeFlag = 1;
            }
        }
    }
}

```

```

    }
}
else
{
    if(distL > maxDistWall)
    {
        cout << "> maxDistWall ";
        if(distL < preDist)
        {
            cout << "center" << endl;
            turn.center();
            usleep(i2cBusWait);
        }
        else
        {
            cout << "extreme left" << endl;
            turn.left(extreme);
            usleep(i2cBusWait);
            extremeFlag = 1;
        }
    }
    else
    {
        if(distL > optDistWall)
        {
            cout << "> optDistWall ";
            if(distL < preDist)
            {
                if(distL > optDistWall + 10 || !extremeFlag)
                {
                    cout << "center" << endl;
                    turn.center();
                    usleep(i2cBusWait);
                }
                else
                {
                    cout << "almost right" << endl;
                    turn.right(extreme);
                    usleep(i2cBusWait);
                    extremeFlag = 0;
                }
            }
        }
        else
        {
            cout << "turn left" << endl;
            turn.left((distL - optDistWall) * 2);
            usleep(i2cBusWait);
        }
    }
    else
    {
        if(distL < optDistWall)
        {
            cout << "< optDistWall ";
            if(distL > preDist)
            {
                if(distL < optDistWall - 10 || !extremeFlag)
                {

```

```

        cout << "center" << endl;
        turn.center();
        usleep(i2cBusWait);
    }
    else
    {
        cout << "almost left" << endl;
        turn.left(extreme);
        usleep(i2cBusWait);
        extremeFlag = 0;
    }
}
else
{
    cout << "turn right" << endl;
    turn.right((optDistWall - distL) * 2);
    usleep(i2cBusWait);
}
else
{
    cout << "opt center" << endl;
    turn.center();
    usleep(i2cBusWait);
}
}
preDist = distL;
}
}
else
{
    go.stop();
    usleep(i2cBusWait);
    cout << "stopping ..." << endl;
}
} // process

```

```
/*
Michael Simmons
Master Project:
  gRAPI: an indoor four-wheeled robot API for a gumstix connex

Last Modified: 17 Feb 2009

Header file for RSnsr class

The RSnsr Class extends the Srf10 class
*/

#ifndef RSNSR_H
#define RSNSR_H

#include "Srf10.h"

class RSnsr : public Srf10
{
private:
  int dist1;
  int dist2;
  int low, mid, high;
  int curDist;
  bool oldFlag;

public:
  RSnsr(int handle, int addr, char rt = 'c' );
  int mReadS();
}; // RSnsr

#endif
```

```

/*
Michael Simmons
Master Project:
  gRAPI: an indoor four-wheeled robot API for a gumstix connex

Last Modified: 17 Feb 2009

RSnsr class

The RSnsr Class extends the Srf10 class
*/

#include "RSnsr.h"

RSnsr::RSnsr(int handle, int addr, char rt) : Srf10(handle, addr, rt)
{
  dist1 = 0;
  dist2 = 0;
  oldFlag = false;
} // RSnsr

int RSnsr::mReadS()
{
  int dist;

  dist = readS();
  mid = dist1;
  if(dist2 >= mid)
  {
    high = dist2;
    low = 0;
  }
  else
  {
    low = dist2;
    high = 0;
  }

  curDist = (dist >= mid)?
    ((high)?((dist >= high)?high:dist):mid):
    ((low)?((dist >= low)?dist:low):mid);
}

/*
  if(dist >= mid)
    if(high)
      if(dist >= high)
        curDist = high;
      else
        curDist = dist;
    else
      curDist = mid;
  else
    if(low)
      if(dist >= low)
        curDist = dist;
      else
        curDist = low;
    else

```

```
                curDist = mid;
*/

    if(oldFlag)
        dist2 = dist;
    else
        dist1 = dist;

    oldFlag = !oldFlag;

    if(curDist)
        return curDist;
    else return dist;
} //mReadS
```