SELF-GUIDED ROBOT VEHICLE PLATFORM

ECET 49100—Senior Design Project, Phase II

Presented in Partial Fulfillment of the Requirements

for the Degree Bachelor of Science in

Electrical Engineering Technology

By

Jennifer Vacendak

And

Khalid Al Marri

Purdue University Calumet

Department of Engineering Technology

Electrical & Computer Engineering Technology Program

Spring Semester 2011

ECET Advisor Signature: _____

Advisor Printed Name: _____

Course Coordinator: Professor Omer Farook

ACKNOWLEDGEMENTS

First and foremost, we would like to recognize our ECET Advisor and Phase II Course Coordinator, Professor Omer Farook. Without his encouragement, thoughtful suggestions, and careful attention to progress, lack of motivation would surely exist. In part, we were able to complete our project on time and see our vision through to the end because of his contributions. We would also like to recognize Professor Masoud Fathizadeh for his Phase I contributions.

Without prior knowledge in specific subject areas, this project would be nearly impossible to complete in the given timeline. The following is a list of the courses that aided our work:

COM 30700: Written & Oral Communication for Engineers
ECET 10000: Introduction to Electrical & Computer Engineering Technology
ECET 10200: Electrical Circuits I
ECET 10900: Digital Fundamentals
ECET 20900: Introduction to Microcontrollers
ECET 29600: Electronic System Fabrication
ECET 45500: C++ Object Oriented Programming
ECET 45600: Computer Hardware Design
ECET 49000: Senior Design Project Capstone Course, Phase I
ECET 49100: Senior Design Project Capstone Course, Phase II
ENGL 22000: Technical Report Writing
IET 30800: Engineering Project Management & Economic Analysis

We would also like to acknowledge financial sponsors of the project. As recipients of both the Undergraduate Research Grant and LSAMP (Louis Stokes Alliance for Minority Participation) Award, we are appreciative toward the providers so that we may be less burdened with project expenses.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

ABSTRACT

Automotive safety has always been a top priority, as evident with the adoption of the seat belt and the airbag. Automotive safety can be limited by lack of technology, however. Recently society has begun to see a slew of abbreviations with regard to automobile features: ABS, ASR, ESC, ICRS, EBA. These terms may seem like marketing jargon at first, but the technology behind them can actually prevent accidents and protect vehicle occupants from harm.

This project aims to provide information on the different automotive safety technologies in use today, uncover technologies that may be implemented in the future, put some of these ideas into practice in a small-scale environment, shed light on the importance of automotive safety, and inspire others to continue research in this field.

This project follows research performed by Volkswagen (as well as other major vehicle manufacturers) and investigates the technologies applied in the various DARPA (Defense Advanced Research Projects Agency) competitions for automated vehicles (in addition to other related projects).

A pre-built robotic platform is programmed to behave like an automated vehicle. It demonstrates two programs: in the Main Program, the robot navigates itself to a predetermined location on a physical map, using select sensors to detect and avoid objects, and obey a traffic light. In the Lane-Keeper Program, the robot maintains its position within a lane and adjusts its speed to obey a speed limit. All project details are planned with scalability in mind, so that the concepts can be adapted to larger applications (i.e. an automobile in the real world).

I. INTRODUCTION

Traditionally, automobiles have been predominantly mechanical in nature. Over the years, more and more electrical components have been introduced. Cars are much more "aware" of their individual actions and parts and can communicate better with their drivers. Some of these advancements have been geared toward safety. Sensors can monitor conditions, gather information, and direct the car to protect its occupants from harm in the event of a collision, or correct itself when presented with dangerous circumstances. According to the National Highway Traffic Safety Administration (NHTSA), there were more than 41,000 vehicle-related deaths and nearly 2.5 million automobile injuries in the United States in 2007. Technology, in effect, makes cars safer and can help reduce this statistic.

This paper studies the different automotive safety technologies in use today, uncovers technologies that may be implemented in the future, and sheds light on the importance of automotive safety. It also delves into the intimate details of the Senior Design Project, entitled *Self-Guided Robot Vehicle Platform.*

The purpose of the project is to showcase some of these technologies via a robotic vehicle in a small-scale environment. By doing so, an understanding of how a vehicle interacts with its world is achieved, which is the primary goal. Secondary objectives include the following:

- Practicing electronics fabrication techniques
- Troubleshooting component functionality
- Writing and debugging code
- Programming techniques for specific hardware
- Effective time and project management
- Adapting to new software tools
- Planning and constructing a physical map
- Identifying pitfalls and working around them
- Teamwork and cooperation

This project is unique in the sense that it demonstrates a technique for an automated vehicle to obey traffic signals (speed limits, traffic lights, or stop signs) wirelessly. Since outfitting an actual vehicle with electronic equipment is beyond the scope of the Senior Design Project Capstone Course (and the budget), sensors and logic are added to a pre-built robot platform. The robot will need to use both to navigate to a pre-determined location on a physical map, avoiding obstacles and obeying traffic laws along the way.

The project's aim is to prove how advancements in automotive safety and automated vehicles can directly affect people's daily lives and protect them. Upon full-scale implementation, the project's concepts and their final product may be suitable for possible use with automobile manufacturers' Research & Development teams and, perhaps eventually, for consumers who wish to drive a safer car.

II. BACKGROUND

Automotive safety is not a new topic by any means. In fact, seat belts date back to the 1800s.[5] The first electromechanical airbag system was invented in 1968.[4] It doesn't stop there, though. Cars today host a myriad of high-tech safety features, and there are more to come.

1. CURRENT & DEVELOPING SAFETY TECHNOLOGIES

Volkswagen, for example, has taken great strides in developing technology for safety applications. Below are some active safety highlights:[20]

- Engine Braking Assist: Reduces skidding on slippery surfaces.
- Electronic Differential Lock: Permits smooth starts on uneven surfaces.
- Hydraulic Brake Assistant: Helps to shorten braking distance.
- Electronic Stabilization Program: Generates corrective forces during sudden accident avoidance maneuvers to help maintain control.
- Anti-Lock Brake System: Enables steering control while braking.
- Electronic Brake-Pressure Distribution: Maximizes braking power.
- Anti-Slip Regulation: Helps maximize loss of wheel traction.
- Lane Assist: Monitors traffic lines on the pavement. A vibration in the steering wheel warns if the vehicle has drifted out of a lane without the driver signaling the lane change.
- Adaptive Cruise Control: Monitors the distance to vehicles in front via radar. When cruise control is engaged, the feature automatically reduces and increases speed to maintain a safe preset distance.
- Side Assist System: Monitors rear and side blind spots. If another vehicle is identified, the system informs the driver via LEDs in the exterior mirrors.
- Dynamic Adaptive Headlights: Sweep around to follow curves. Adaptive headlights pivot up to 15 degrees, clearly illuminating the road and possible obstructions.
- Back-Up Camera: Makes parking and reversing easier. A real-time view of the rear parking area is shown on the navigation screen.
- Voice Activation: Offers a more personalized way to interface with the vehicle's important features in a hands-free mode.

The same piece of literature identified the following as future safety technologies:

- Engine Braking Control: Detects the loss of wheel adhesion, comparing the speed of the drive wheels with that of free-moving wheels, accelerating slightly if necessary.
- Automatic Emergency Brake: Goes a step further than the ESP and executes a full braking maneuver to help prevent a collision, or to greatly reduce the speed of a collision, as soon as the distance to the vehicle in front reduces so much that a crash is inevitable.
- Automatic Distance Control: Related to the ASB, this feature brings the vehicle to a complete standstill, if necessary, behind a car ahead. This feature also provides a

collision warning and a predictive distance recognition that independently brakes the car if traveling too fast.

- Attention Control: A camera in the cockpit monitors the driver's eyelids, recording the duration and frequency of blinking. At the first signs of fatigue, the driver is immediately warned acoustically and requested to take a break.
- All-Round Vision: Uses all sensor data or "surroundings perceptions" to help detect possible hazards and activate all necessary crash prevention measures. Related Enhanced Night Vision uses infrared sensors to recognize pedestrians and their position or distance from the vehicle and notifies the driver.
- Road Sign Display: Registers speed limits along the roadway and displays them digitally in the driver's cockpit.
- Parking Assist: For everyday use, assists or automatically maneuvers the vehicle into any parking space. Sensors record the dimensions of the space and any obstacles. The ideal route is calculated. By pressing a button, the driver activates the parking assistant and the car reverses into the space.

Volkswagen operates an Electronics Research Lab in California to continuously improve its automotive technology in many areas. Volkswagen is a major player in the field of safety, but by no means the only one. Many major car manufacturers (especially luxury brands) are also touting cutting-edge safety features (such as Mercedes-Benz, Lexus, BMW, Volvo, and Ford, just to name a few).

While many manufacturers are updating technology on manually-driven automobiles, automated vehicles are up-and-coming. There are several benefits to this:

- Eliminates human error, the leading cause of accidents
- Prevents crashes and, when a crash is unavoidable, protects the occupants
- Reduced emissions
- Allows certain groups of individuals who otherwise cannot drive (the blind and elderly, for example) to be more mobile

2. RECENT EXPERIMENTAL PROJECTS

There are several notable projects in the field of automated vehicles. Below is a brief description of each.

a) DARPA Grand Challenges

DARPA (Defense Advanced Research Projects Agency) has spurred the development of autonomous robotic vehicles. In previous years, it has issued challenges to spark research and development in the area of unmanned ground vehicles. One such challenge was the Grand Challenge 2005, which required teams to build an autonomous robotic vehicle to navigate over a treacherous 131.2-mile course in the Mojave Desert.[6]

Stanley, a robotic VW Touareg that was collaborative effort between Stanford University and VW's Electronics Research Lab,[3] won this challenge. Stanley was

outfitted with GPS, an inertial measurement unit, four laser rangefinders, a radar system, stereo camera pair, and a monocular vision system.[15]

The runner-up, Carnegie Mellon University's Sandstorm, featured four LIDAR units, a radar unit, an inertial navigation system, GPS, and a pair of cameras for stereo vision. An updated version of the vehicle has seven LIDAR units, and short- and long-range radar.[13]

Urban Challenge was held in 2007 in Victorville, CA. This challenge required teams to build an autonomous robotic vehicle to navigate through traffic on a closed course. This demanded performance in areas such as merging, parking, passing, and negotiating intersections with a stop sign.[6]

Junior, Stanford's VW Passat and Stanley's successor, contained such technology as a multiple LIDAR (Light Detection and Ranging) sensors for 3D imagery and lane detection, an inertial measurement unit, five radar sensors, a six-way camera system, and an Applanix GPS set (35mm accuracy). Junior took second place at this event.[14]

The winner was Boss, a Chevy Tahoe from Carnegie Mellon University. Its equipment set contained many LIDAR units of different types and Applanix GPS.[16]

b) Google Automated Car

For this project, the best and brightest engineers were gathered from the DARPA Grand Challenges. Google employs a Toyota Prius as its autonomous vehicle, geared with video cameras, radar sensors, laser rangefinders, detailed real-world maps gathered from manually-driven vehicles (including lane markers and traffic signs). This means the software in the Prius is already familiar with the environment it's about the encounter.[17] A team of two humans monitor the navigation and stand by, ready to take control of the vehicle if necessary. Astonishingly, Google's self-driving cars have logged over 140,000 miles.[11]

c) VisLab Intercontinental Autonomous Challenge

This challenge ran from July 2010 to October 2010, and involved four driverless vehicles (with little to no human intervention) traveling from Parma, Italy to Shanghai, China. This is interesting because there were areas traveled where no map was available. That being said, the first vehicle drove autonomously in select sections and its primary goal was to collect sensor data. The second vehicle followed the leader either visually or with GPS waypoints. It required no human intervention. The other two were backup vehicles. The team traversed such conditions as Moscow traffic.[18]

d) VW Golf GTI 53+1

VW retains the firm belief that driving (by humans) should be fun. However, the company also has a firm commitment to safety and advancing technology. On VW's

own test track in Wolfsburg, Germany, the GTI 53+1 can reach peak performance while driving automatically. In fact, the car can achieve course times on par with professional drivers! The GTI's primary equipment consists of a high-accuracy DGPS (Differential GPS) and laser scanner mounted up front.[19]

e) Audi's Shelley

Audi designed a similar concept to the GTI: a TTS named Shelley. This car, a collaborative effort among Audi, Stanford University, VW's ERL, and Oracle, conquered Pikes Peak in September 2010 without a driver, without stopping, in 27 minutes. It features Java programming and DGPS.[2]

f) Volvo's Road Train

Volvo, in conjunction with Safe Road Trains for the Environment (a European Commission project), developed what is referred to as a "road train." This system is a semi-autonomous convoy in which the lead vehicle (driven by a human) is followed by computer-controlled cars. Cameras are responsible for keeping track of each car in front. This project was an effort to decrease emissions and allow drivers (in the cars that follow the leader) to relax and focus on other tasks.[12]

g) Ford's Smart Intersection

According to a recent article by Ford Motor Company, "The smart intersection, established near the Ford Research & Innovation Center in Dearborn, Mich., communicates with specially equipped test vehicles to warn drivers of potentially dangerous traffic situations, such as when a vehicle is about to run through a red light. The intersection is outfitted with technology that can monitor traffic signal status, GPS data and digital maps to assess potential hazards, and then transmit the information to vehicles."[7] It is unknown which wireless technology is being implemented.

III. SYSTEM DESCRIPTION

This project is based on the iRobot Create® platform, which takes instructions from the onboard Command Module (microcontroller). It can also use infrared to detect objects (using a Rangefinder), receive outside signals (via a receiver located on top of the robot), and follow a line (two Cliff Sensors under the robot). Figures 1 and 2 below diagram the main robot components, and Figures 3 and 4 show how the parts interact for each program.

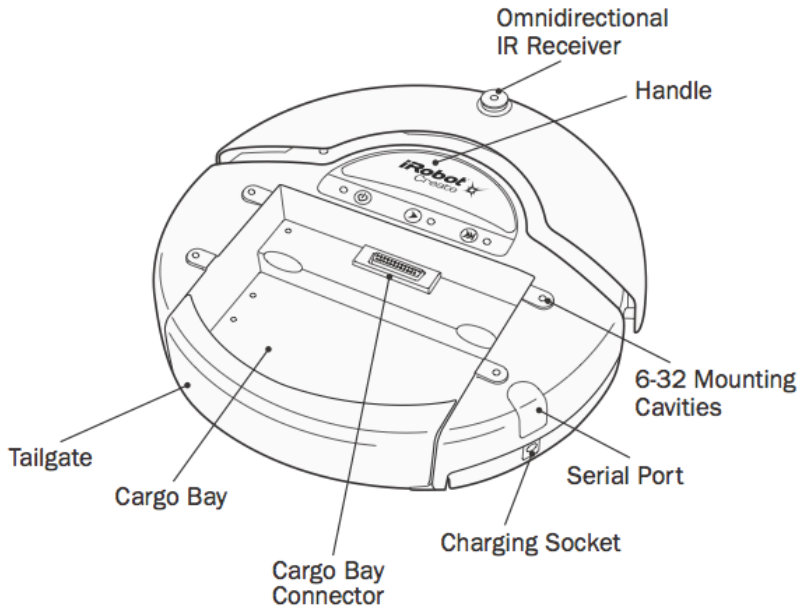


Figure 1: iRobot Create®, top view (image courtesy of iRobot)

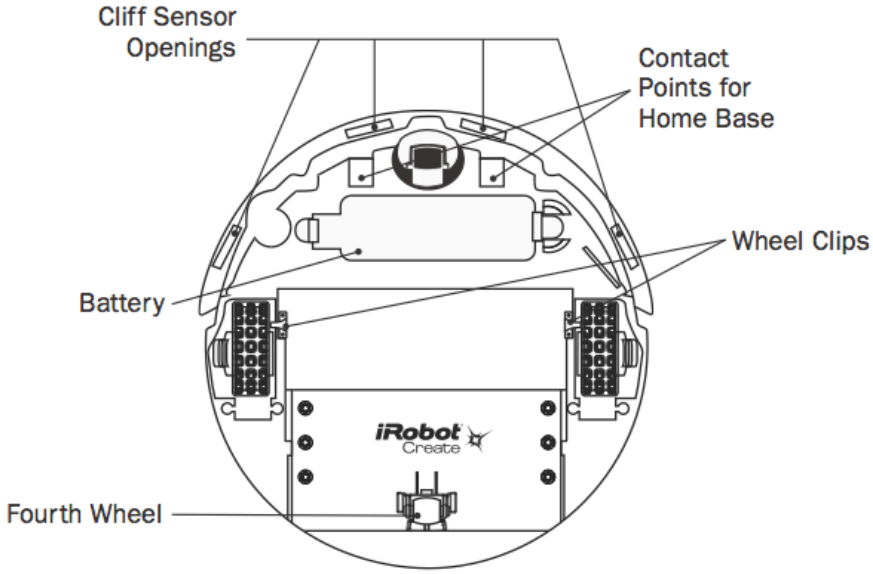


Figure 2: iRobot Create®, bottom view (image courtesy of iRobot)

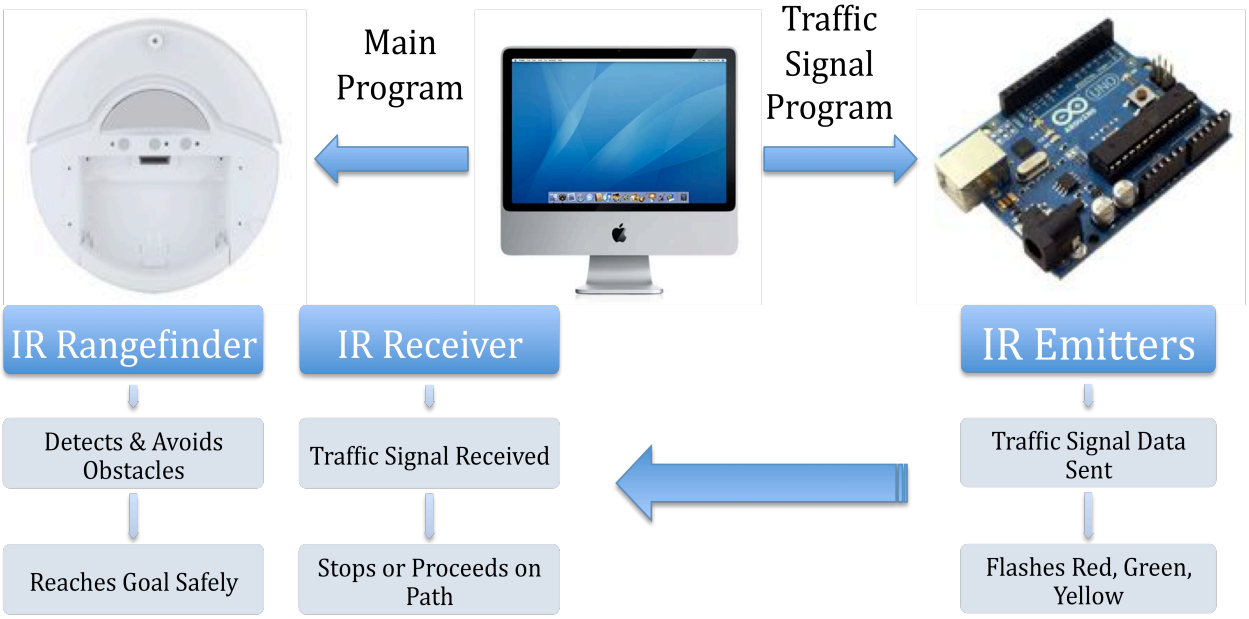


Figure 3: Diagram outlining hardware functions in Main Program

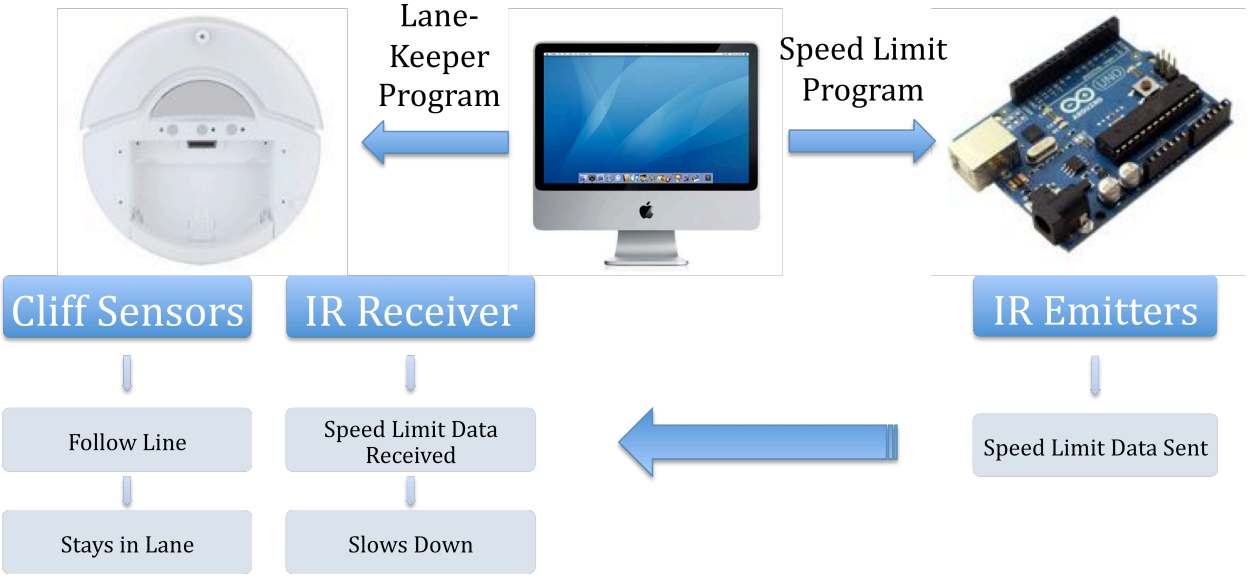


Figure 4: Diagram outlining hardware functions in Lane-Keeper Program

The following sections describe the system hardware and software, provide analogies to their real-world counterparts, and justify why these specific items were chosen for this project. For further details, the datasheets, schematics, and code can be found in Appendices A, B, and C (respectively).

1. SYSTEM HARDWARE

This system features three main hardware categories: robot platform, microcontrollers, and sensors. Below is a detailed description of each.

a) Robot Platform

At the heart of the project is an iRobot Create®. This is a preassembled robot platform, much like the iRobot Roomba® robot vacuum, but for developers. It features the following:[9]

- Ability to understand individual commands or scripts sent from a PC over a serial cable
- Over 30 built-in sensors
- 25-pin expansion port for adding a Command Module (microcontroller) or additional hardware
- Cargo bay for mounting additional hardware

b) Microncontrollers

The iRobot Create® Command Module houses a microcontroller, which allows one to expand the robot's programming capabilities and autonomy. It has the following specifications:[9]

- 8-bit 18MHz Atmel® ATmega168 Microcontroller with 16KB Flash memory
- 4 expansion ports for adding hardware
- 2 analog only inputs
- 4 digital only I/O pins
- 6 digital I/O or analog input pins
- Ability to communicate with a PC over USB

Additionally, two Arduino Uno boards are present in this category. These are based on the ATmega328 microcontroller. These boards feature the following:[1]

- 5V operating voltage
- 14 digital I/O pins [6 provide Pulse Width Modulation (PWM) output]
- 6 analog input pins
- 32KB Flash memory
- 16MHz clock speed

c) Sensors

This project utilizes several sensors. Here is a list and brief description of the sensors:

- Cliff Sensors
    - Infrared (IR) Emitter & Receiver Pair
    - Built into robot
    - Used for line detection in this project
- Sharp IR Rangefinder
    - Takes a distance reading and returns an analog voltage
    - Mounted on a servo for rotation
    - Used for obstacle detection and avoidance
- IR Receiver
    - Built into robot
    - Detects external IR signals (such as from a remote control)

2. SYSTEM SOFTWARE

In order to program the hardware, software is needed. Below is a list and brief description of software included in the project.

- WinAVR
    - Programmer's Notepad
        - Text editor/Integrated Development Environment (IDE) for compiler
        - Interfaces directly with Command Module
    - MFile
        - Makefile generator
- Bloodshed Dev C++
    - Free C/C++ IDE
- Arduino
    - For writing code and uploading to Arduino board

3. HARDWARE AND SOFTWARE CHOICE JUSTIFICATION

In order to understand the reasons why specific hardware and software were chosen for this project, justification is provided. This project could have taken place on a larger scale, but there are several reasons why it did not. The primary reason is because cars are expensive. The amount of research, equipment, time, and money needed to get a project like that completed is far beyond the scope of the Senior Design Project Capstone Course.

Cars can also be quite dangerous if they are not interfaced with properly. For example, if a small robot tears off into a wall because of a programming mistake, no damage is done; this is not the case with a full-size car. The project team possesses C/C++ programming knowledge of microcontrollers and personal computers, but they have never interfaced with a car.

Budget was also the main deciding factor in choosing sensors. Advanced rangefinders and IR localization systems could have been implemented to increase accuracy, but laser rangefinders and Hagisonic StartGazers are much too expensive (starting at $1,000 per sensor). A camera would have been a nice addition for obstacle detection and avoidance (and within budget), but the microcontroller in the Command Module most likely would not have enough memory.

Another deciding factor was scale. Since the project takes place in a 7'x7' area, the decision was made not to use a Global Positioning Sensor (GPS). Simply put, accuracy to within meters is not accurate enough for the project scale.

As for software, hardware manufacturers' advice was taken. iRobot recommended WinAVR, Arduino has its own software, and Bloodshed Dev C++ was recommended by a developer using the iRobot platform.

4. ANALOGIES TO REAL-WORLD COMPONENTS

In order to better convey the idea that the project simulates an automated vehicle, here is how each major component relates to objects in the real world:

Table I. Analogous Systems

Project Component	Analogy
Robot	Automated car
Command Module	Engine/Main Computer
Cliff Sensor	Lane-Keeping Device
Sharp IR Rangefinder	Obstacle Detector
IR Receiver	Traffic Signal Receiver
Arduino Boards	Traffic Signal Transmitters
Physical Map	Real-World Environment
Boxes on Map	Obstacles; Black=Known, Green=Unknown
White Tape	Lane Marker

5. THE WAVEFRONT ALGORITHM

The main parts of this project include storing a map, updating it if conditions change, detecting and avoiding obstacles, and finding a solution to reach the goal location. These are all parts of the Wavefront Algorithm. Wavefront is an open-source algorithm that aids in robot mapping and navigation. Listed on the next page are the main ideas behind the algorithm.[8]

1. Create an X-Y grid and define open space, obstacles, robot starting location, and goal location.
2. A path (or several) is calculated based on the goal location and open spaces around it.
3. The robot uses its rangefinder to detect obstacles that were not defined in the original map. It then updates the map, recalculates a new path, and moves around the obstacle.
4. If a solution to locating the goal is not found, the map will be reset and the robot will begin looking for obstacles again.
   In the figure below, the black squares represent known obstacles, and the gray squares represent obstacles found with the rangefinder. *R* is for Robot, and *G* is for Goal.

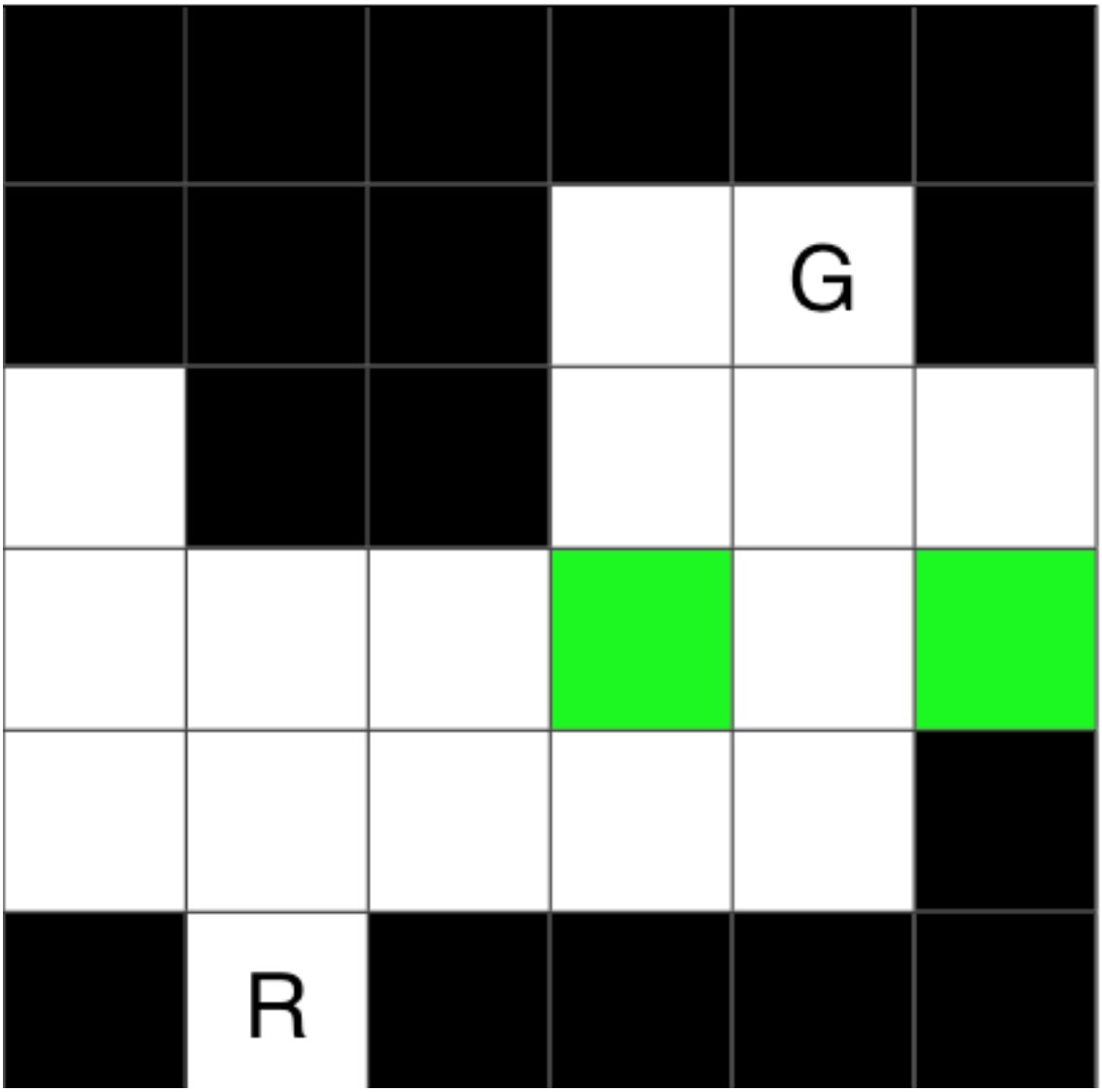


Figure 5: Robot's view using Wavefront

6. THE MAIN PROGRAM

This is the portion of the project that incorporates the Wavefront Algorithm, IR rangefinder, Arduino "traffic light," and IR receiver. The figure that follows explains the logic behind the program.

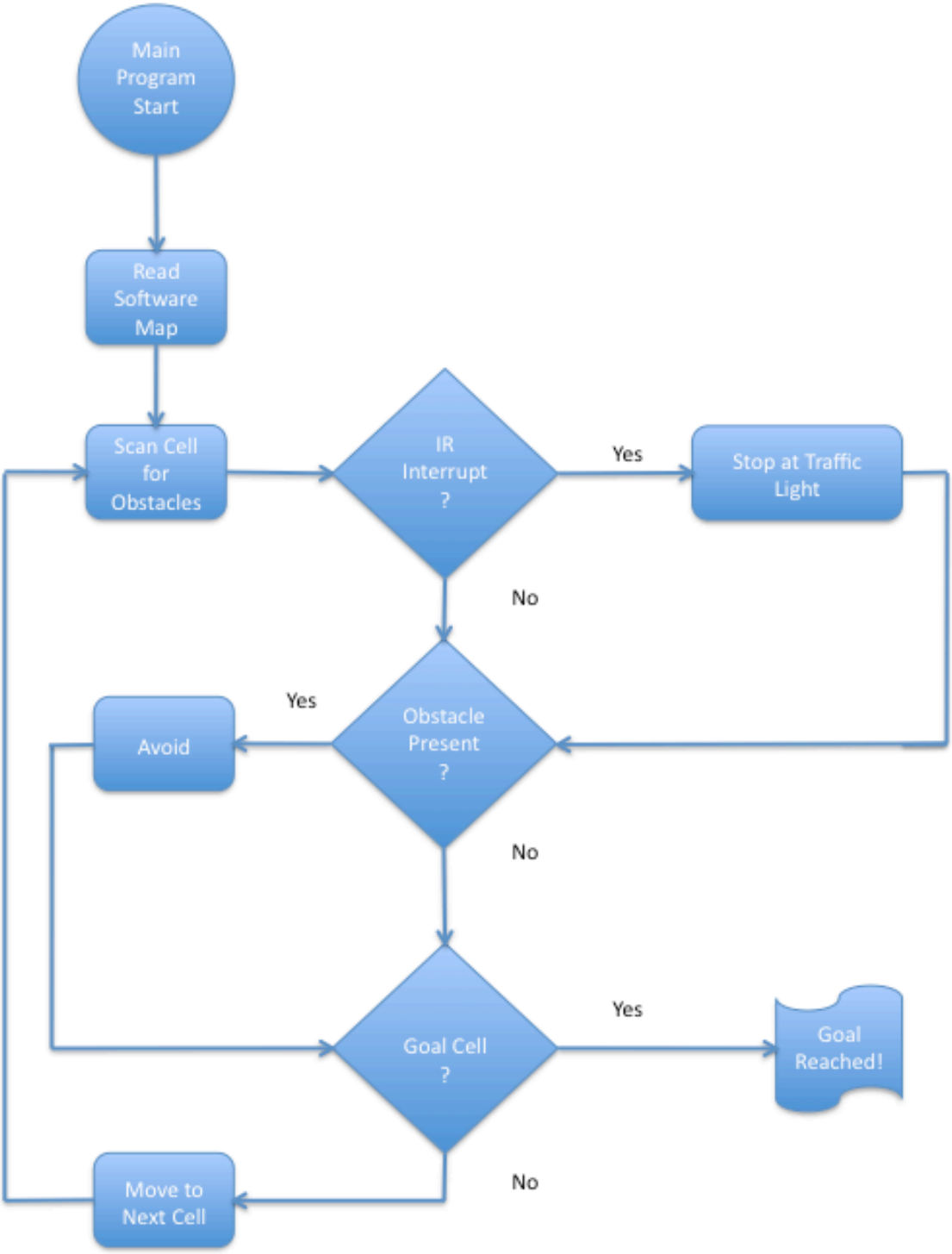


Figure 6: Flowchart for Main Program

7. THE LANE-KEEPER PROGRAM

This is the portion of the project that incorporates the Cliff Sensors, Arduino "speed limit sign," and IR receiver. The figure that follows explains the logic behind the program.

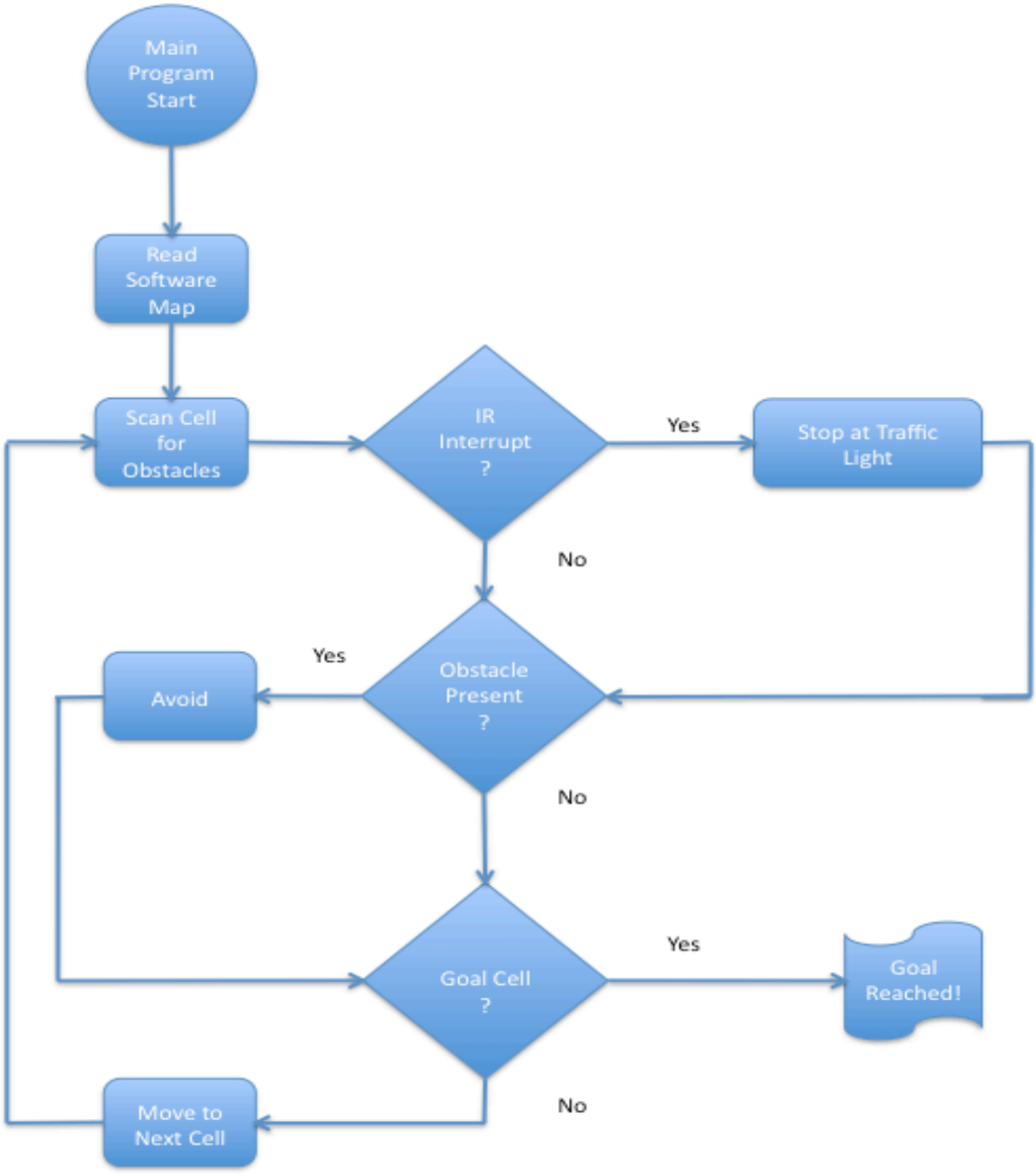


Figure 7: Flowchart for Lane-Keeper Program

IV. PROBLEMS ENCOUNTERED

As with any project, setbacks are bound to occur. This section outlines the problems that were encountered during the course of the project, as well as possible problems associated with automated vehicles in the real world.

1. PROJECT PROBLEMS

The largest hurdles to overcome were that of knowledge and experience. The project team needed to adapt to all of the new software and a robot platform they had never interacted with before. With a new platform comes new vocabulary for programming, which also needed to be learned. Even tasks that the team members have experience with (soldering, writing code, etc.) took additional time because they do not perform these tasks frequently.

Once they became comfortable using the software and platform, parts procurement became another issue. In the first set of parts, some items were missing, and some things were extra. This was most likely due to miscommunication between team members. Upon ordering the Arduinos, it became apparent that they were on backorder. A second order with a different vendor was placed, and it was resolved.

Another parts issue (also tied to unfamiliarity with the platform) occurred when a Lynxmotion Line Follower was ordered but not necessary. It performs the same function as the Cliff Sensors in the robot, so those were used instead.

Programming shortcomings were present as well. For instance, lane markers are located to the left of cars driving on a road. In an attempt to emulate this, the robot was initially programmed to use its Left Cliff Sensor to detect the line. After much trial and error and limited success, the team scrapped this idea and used the Left Front and Right Front Cliff Sensors to center on the line. It was not ideal, but it still demonstrates the same technology.

The largest disappointment occurred when the team realized the Lane-Keeper program could not join the Main Program. The robot can accurately track a line running down the middle of it, which means each grid cell would need to be a lane with a line running down the middle (instead of to the left). The robot would need to hop across two cells to be out of "oncoming traffic." This would make the Main Program and map much more complicated than it needs to be. It simply did not make sense to have both programs running simultaneously on such a small scale, even though a one-program solution was what the team was aiming for originally.

2. REAL-WORLD PROBLEMS

The primary issues with regard to automated vehicles are weather and other types of interference. Rain and snow can skew sensor readings (especially vision), and interference can degrade wireless communications. Another issue is wireless security.[10]

V. EXPERIMENTAL RESULTS

The Wavefront Algorithm included a handy tool for simulation. This is provided so one does not create a map, upload it to the robot, watch it play out, and no solution is found to the goal location. The simulator saves time by running the map and providing an answer within seconds. Here are some different maps that were run through the simulator (0=free space, W=obstacle, G=goal, R=robot):

```
Finished Wavefront:
W W W W W W
0 W W R G W
0 W W 0 0 0
0 0 0 0 0 W
0 0 0 0 0 W
0 W W W W W

steps: 44
```

```
Finished Wavefront:
W G R W W W
W 0 0 0 0 W
0 W W 0 0 W
0 W W 0 0 0
0 0 0 0 0 0
W W W W W 0

steps: 36
```

```
Finished Wavefront:
W 0 0 W W 0
0 0 0 W W 0
W 0 0 0 0 0
0 0 W 0 W 0
0 0 0 W 0 R
W 0 W W W G

steps: 79
```

```
Finished Wavefront:
W 0 0 W W W
0 0 0 W W G
W 0 0 0 0 R
0 0 W 0 W 0
0 0 0 W 0 0
W 0 W W W 0

steps: 54
```

```
Finished Wavefront:
W 0 0 W W W
0 W 0 W R G
0 0 0 0 0 0
0 0 0 0 W 0
0 0 0 0 0 0
W 0 W W W 0

steps: 47
```

```
Finished Wavefront:
G R 0 W W 0
0 0 0 W W 0
W 0 0 0 0 0
0 0 0 W W 0
0 0 0 W 0 0
0 0 W W W 0

steps: 24
```

Figure 8: Wavefront simulator results

VI. CONCLUSION

1. PROJECT DISCUSSION

Although we are not the first to use the iRobot Create® platform, study robotic vehicles, or attempt to create one, our research and findings have led to a new perspective and appreciation for automotive safety and automated vehicles. Our goal was not to invent something new. We wished to learn about the technology behind automobiles (present and future) and put some of those features into action to better understand them and how they can make our roadways safer. By doing so, it served as an excellent exercise for learning and putting our skills to work. In that sense, the project is a success. By documenting the project, we hope we have educated others about safety technology and automated vehicles and provided inspiration to continue work in those areas.

2. FUTURE IMPROVEMENTS

If we were to make changes to the project to make it more realistic, we would do the following:

- Choose a different processor with more memory. This would allow for larger maps and memory-hungry sensors.
- Try our hand at a vision system. Although most major automobile manufacturers rely primarily on infrared, vision has been shown to work (Stanley, DARPA Grand Challenge).
- Add more sensors all around the robot. Currently it can only "see" the front and part of the sides. This would be helpful for adding reverse/parking features.
- Incorporate movement while scanning. At this point, the robot pauses between cells to detect obstacles in the next cell. A real-world car would need to detect things on-the-fly.
- Increase the map size and make it more realistic (trees, buildings, wildlife that dashes out, other vehicles, pedestrians, day and night conditions, etc.).

Scaling up has been mentioned, and this is where we address it. Based on our research, here are the primary sensors we would choose if we built a full-size automated vehicle:

- DGPS for its high accuracy
- Inertial measurement unit for areas where GPS is unavailable
- LIDAR sensors (various types and ranges) for vision

We would also include maps from other vehicles, when that technology arrives officially. If each vehicle (autonomous or not) was outfitted with a vision system and could record its route, other vehicles could download the most recent information prior to beginning a trip. This would prepare the autonomous vehicle for conditions ahead, instead of adapting itself on-the-fly when things crop up unexpectedly.

In addition, IR is not suitable for real-world applications of road signs and traffic signals (due to line-of-sight and range limitations). A wireless protocol that does not exhibit these limitations would need to be implemented.

VII. COGNITIVE UNDERSTANDING OF STUDENT LEARNING OBJECTIVES

The Electrical Engineering Technology program must demonstrate that graduates have:

a. an appropriate mastery of the knowledge, techniques, skills and modern tools of the discipline of Electrical Engineering Technology;

b. an ability to select and apply current knowledge and adapt to emerging applications of mathematics, science, engineering and technology to Electrical Engineering Technology problems that require application of principles and applied procedures or methodologies;

c. an ability to conduct standard tests and measurements; to conduct, analyze and interpret experiments; and to apply experimental results to improve processes;

d. an ability to design systems, components or processes for broadly-defined Electrical Engineering Technology problems appropriate to program educational objectives;

e. an ability to function effectively as a member or leader on a technical team;

f. an ability to identify, analyze and solve broadly-defined Electrical Engineering Technology problems;

g. an ability to apply written, oral, and graphical communication in both technical and nontechnical environments; and an ability to identify and use appropriate technical literature;

h. an understanding of the need for and an ability to engage in self-directed continuing professional development;

i. an understanding of and a commitment to address professional and ethical responsibilities including a respect for diversity;

j. a knowledge of the impact of engineering technology solutions in a societal and global context;

k. a commitment to quality, timeliness, and continuous improvement.

It was requested that the list above be addressed point by point (minus point *i*), with respect to the project.

a. Knowledge of basic electrical and digital concepts, C/C++ programming, microcontrollers, electronics fabrication, project management, and technical report writing are applied.

b. The research is taken from current sources, referring to present and future technologies. Choices for scaling the project up are based on this current information.

c. The Wavefront simulation software and its results aided in choosing a map that would suit our purpose.

d. Each sensor and hardware component of the system was hand-picked with the broadly-defined problem of automotive safety in mind. The concepts used here can easily be scaled to the real world.

e. Our team consisted of two individuals, as well as a Project Advisor. Each role was clearly defined, and the team functioned well as a whole during each project phase.

f. This project served as a research opportunity, rather than a problem to be solved. The broadly-defined problem of automotive safety was identified and analyzed, but not necessarily solved in its entirety. The team did, however, solve issues surrounding road signs and traffic signals.

g. The presentation and report portions of this project serve as effective communication to both technical and nontechnical audiences. Technical reports were viewed and studied prior to writing this report to ensure proper formatting. Some also aided in research.

h. The project presentation and report inspire professionalism in their own right. The team understands the importance of conveying ideas and retaining respect when presenting the project.

j. The entire project was based on societal and global impact. Automotive safety technology has many very beneficial impacts (such as preventing accidents and saving lives). The road sign/traffic signal solution provides for safety when drivers are distracted or breaking the law.

k. This project displayed a commitment to quality in its presentation. Decisions were made in favor of quality (components, materials), even if it meant a higher cost. Timeliness was difficult to adhere to, but the project was completed on time. Continuous improvement was made during each phase of the project, but there are no plans to continue improvement after May 2011.

VIII. REFERENCES

[1] Arduino. *Arduino Uno.* http://arduino.cc/en/Main/ArduinoBoardUno.

[2] Audi. *Autonomous Audi TTS Pikes Peak Achieves This Year's Goal.* http://www.audiusa.com/us/brand/en/about/main/awards.detail.2010~11~audi_autonomous_tts_pikes_peak_achieves_goal.html, Nov. 2010.

[3] *Autonomous Driving*. http://www.vwerl.com/research/story/Autonomous-Driving.

[4] Bellis, Mary. *The History of Airbags.* http://inventors.about.com/od/astartinventions/a/air_bags.htm.

[5] Bellis, Mary. *The History of Seat Belts.* http://inventors.about.com/library/inventors/bl_seat_belts.htm

[6] DARPA. *Welcome.* http://archive.darpa.mil/grandchallenge/index.asp.

[7] Ford. *Ford Smart Intersection 'Talks' to Help Reduce Accidents, Collisions and Fuel-Wasting Congestion.* http://media.ford.com/article_display.cfm?article_id=28611.

[8] *How to Build a Robot Tutorial – Society of Robots.* http://www.societyofrobots.com/programming_wavefront.shtml.

[9] *iRobot: Robots: Programmable: iRobot Create® Premium Development Package.* http://store.irobot.com/product/index.jsp?productId=2591901&cp=2174940.2591511&s=D-StorePrice-IRBT&parentPage=family.

[10] Koscher et. al. *Experimental Security Analysis of a Modern Automobile.* http://www.autosec.org/pubs/cars-oakland2010.pdf.

[11] Markoff, John. *Google Cars Drive Themselves, in Traffic.* http://www.nytimes.com/2010/10/10/science/10google.html?_r=1, Oct. 2010.

[12] Morgan, Patrick. *"Road Train" Technology Could Let You Doze in the Driver's Seat.* http://blogs.discovermagazine.com/80beats/2011/01/18/road-train-technology-could-let-you-doze-in-the-drivers-seat/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+80beats+%2880beats%29, Jan 2011.

[13] Red Team Racing. *Sandstorm.* http://www.cs.cmu.edu/~red/Red/sandstorm.html.

[14] Stanford Racing Team. *All about Junior.* http://cs.stanford.edu/group/roadrunner/pdfs/final_factsheet_junior.pdf, Oct. 2007.

[15] Stanford Racing. *Technology.* http://cs.stanford.edu/group/roadrunner//old/technology.html.

[16] Tartan Racing. *Boss at a Glance.* http://www.tartanracing.org/press/boss-glance.pdf.

[17] Thrun, Sebastian. *What we're driving at.* http://googleblog.blogspot.com/2010/10/what-were-driving-at.html, Oct. 2010.

[18] Vislab. *Vislab's adventure on the Silk road.* *http://vislab.it/Projects/view/32/VisLab%27s_adventure_on_the_Silk_road.*

[19] Volkswagen. *Automatic GTI.* http://www.volkswagenag.com/vwag/vwcorp/content/en/innovation/research_vehicles/automatic_gti.html, Jul. 2006.

[20] Volkswagen Group of America. *Driving. Safety. Powered by Innovation.* http://www.volkswagengroupamerica.com/media/docs/20090323_vw_safety.pdf.

APPENDIX A: DATASHEETS FOR HARDWARE

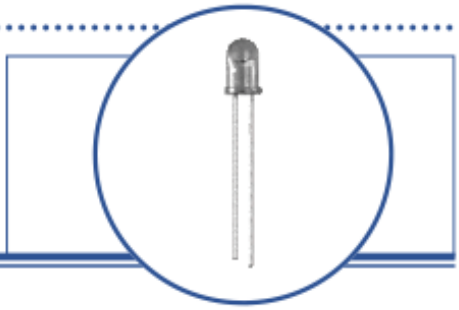
Plastic Infrared Emitting Diode

OP290 Series

TT electronics
OPTEK Technology

Features:

- Choice of narrow or wide irradiance pattern
- Choice of power ranges
- Choice of T-1¾, TO-18 or T-46 package
- Higher power output than GaAs at equivalent LEDs



Description:

Each device in this series, is a gallium aluminum arsenide infrared Light Emitting Diode (LED) that is molded in an IR-transmissive package with a wavelength centered at 890 nm, which closely matches the spectral response of silicon phototransistors, except for OP298 (AA, AB, AC, AD), which has either an 850 nm or 875 nm center wavelength. For identification purposes, each LED anode lead is longer than the cathode lead. ***Package T-1¾*** devices include: **OP290**, **OP291**, **OP292**, **OP294, OP295, OP296**, **OP297**, **OP299 (A, B, C) and OP297FAB,** ***Plastic Package TO-18*** *or* ***TO-46*** devices include: **OP293** and **OP298 (A, B, C, AA, AB, AC, AD)**.

Each **OP290, OP291** and **OP292** series come in three electrical parameters options A, B and C. The **OP290** series forward current is specified under pulse conditions up to 1.5 amps, the **OP291** series forward current is specified under pulse conditions up to 100 milliamps and the **OP292** series forward current is specified under pulse conditions up to 1 amp. The Cathode Lead length is 0.06" (1.52 mm) shorter than the Anode Lead. The silver-copper lead frame offers excellent thermal characteristics.

Each OP293 and **OP298** series come in three electrical parameter options A, B and C. The **OP293** series has an included emission angle of 60° while the **OP298** series has an included emission angle of 25°. The Cathode Lead length is 0.06" (1.52 mm) shorter than the Anode Lead. These devices, which come in a variety of power ranges offering a low cost replacement for TO-18 or TO-46 hermetic packages.

Each OP298 series come with a high irradiance output versions with four electrical parameter options AA, AB, AC and AD. These power options are in the range of **5X** greater than the A, B or C options. The **OP298** series has an included emission angle of 25°. The Cathode Lead length is 0.06" (1.52 mm) shorter than the Anode Lead. These devices, which come in a variety of power ranges offering a low cost replacement for TO-18 or TO-46 hermetic packages.

OP294 and **OP299** are designed for low-current or power-limited applications, such as battery supplies. They are similar to the **OP290** and **OP295**, but use a smaller chip that increases output efficiency at low current levels by increasing current density. Light output can be maximized with continuous (D.C.) forward current up to 100 mA or with pulsed forward current up to 750 mA. The Cathode Lead length is 0.06" (1.52 mm) shorter than the Anode Lead.

Each **OP295, OP296** and **OP297** series come in three electrical parameters options A, B and C. The **OP295** series forward current is specified under pulse conditions up to 5 amps, the **OP296** series forward current is specified under pulse conditions up to 2 amps and the **OP297** series forward current is specified under pulse conditions up to 1 amp. The Cathode Lead length is 0.06" (1.52 mm) shorter than the Anode Lead. The **OP297FAB** has a reversed polarity from the **OP297A, B** or **C**. The silver-copper lead frame offers excellent thermal characteristics. *These devices are UL recognized, File No. S2047.*

All of these devices are spectrally and mechanically matched to the OP593 and OP598 series phototransistors.

Please refer to Application Bulletins 208 and 210 for additional design information and reliability (degradation) data.

Applications:

- Non-contact reflective object sensor
- Assembly line automation
- Machine automation
- Machine safety
- End of travel sensor
- Door sensor
- Battery-operated applications

Pb
RoHS

OPTEK reserves the right to make changes at any time in order to improve design and to supply the best product possible.

OPTEK Technology Inc. — 1645 Wallace Drive, Carrollton, Texas 75006
Phone: (972) 323-2200 or (800) 341-4747 FAX: (972) 323-2396 sensors@optekinc.com www.optekinc.com

Plastic Infrared Emitting Diode

OP290 Series

TT electronics
OPTEK Technology

Part Number Guide — OP290 - OP299 Series

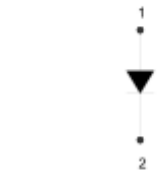
OP 2 9 X X W

Optek Assembly

Photodiode Output Family

Maximum Forward Current
- 0, 5 — 5 amps
- 1, 6 — 2 amps
- 2, 7 — 1 amps
- 3, 8 — 200 milli-amps
- 4, 9 — 750 milli-amps

Electrical Specification Variations:
- A — Parameter A
- B — Parameter B
- C — Parameter C
- D — Parameter D
- AA — Parameter BA
- AB — Parameter BB
- AC — Parameter BC
- AD — Parameter BD



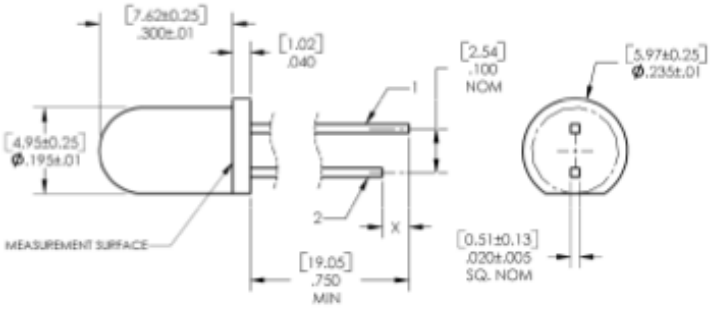
Electrical Connection A, B, C, D

Pin #	LED X=0.060" (1.52 mm)
1	Anode
2	Cathode

Electrical Connection OP297AB

Pin #	LED X=0.060" (1.52 mm)
1	Cathode
2	Anode

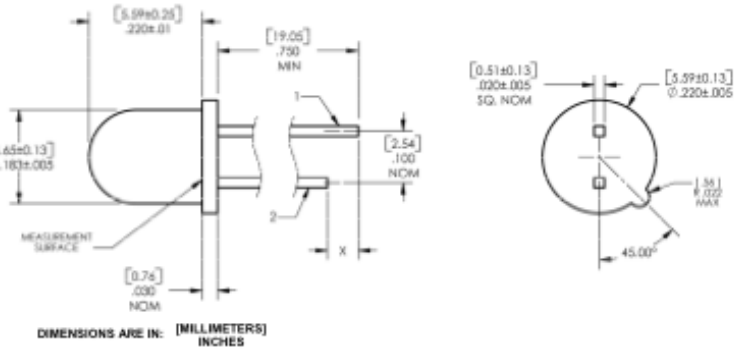
T-1¾ Package
OP290, OP291, OP292, OP294, OP295, OP296, OP297, OP299



Electrical Connection
A, B, C, AA, AB, AC, AD

Pin #	LED X=0.060" (1.52 mm)
1	Anode
2	Cathode

TO-18, TO-46 Package
OP293 & OP298



DIMENSIONS ARE IN: [MILLIMETERS] INCHES

OPTEK reserves the right to make changes at any time in order to improve design and to supply the best product possible.

OPTEK Technology Inc. — 1645 Wallace Drive, Carrollton, Texas 75006
Phone: (972) 323-2200 or (800) 341-4747 FAX: (972) 323-2396 sensors@optekinc.com www.optekinc.com

Absolute Maximum Ratings (T_A=25°C unless otherwise noted)

Storage and Operating Temperature Range	-40° C to +100° C
Reverse Voltage	
OP290, OP292, OP294, OP295, OP297, OP299	5.0 V
OP291, OP293, OP296, OP298	2.0 V
Continuous Forward Current	
OP290, OP291, OP292	150 mA[1]
OP294, OP295, OP299	100 mA[1]
OP295, OP296, OP297	150 mA[1]
Continuous Forward Current, OP293, OP298	
Free Air	100 mA
Board Mounted	133 mA
Full Heat Sink	200 mA
Peak Forward Current	
OP290, OP295 (25 µs pulse width)	5.0 A
OP291, OP296 (100 µs pulse width)	2.0 A
OP292, OP297 (100 µs pulse width)	1.00 A
OP293, OP298 (25 µs pulse width)	2.0 A
OP294, OP299	750 mA

Notes:
1. For OP290, OP291, OP292, OP295, OP296 and OP297, derate linearly 1.67 mA/° C above 25° C (free-air). When used with heat sink (see note 5), derate linearly 2.07 mA/° C above 65° C (normal use). For OP293 and OP298, when measured in free-air, derate power dissipation linearly 1.43 mW/° C above 25° C. For OP294 and OP299, derate linearly 1.80 mW/° C above 25° C.

Absolute Maximum Ratings (T_A=25°C unless otherwise noted)

Maximum Duty Cycle	
OP290 (25 µs pulse width @ 5 A)	1.25%[1]
Lead Soldering Temperature [1/16 inch (1.6 mm) from case for 5 seconds with soldering iron]	260° C[2]
Power Dissipation, Free Air	
OP290, OP291, OP292, OP295, OP296, OP297	333 mW[3]
OP293, OP298	142 mW[3]
Power Dissipation, Board Mounted	
OP290, OP291, OP292, op295, OP296, OP297	533 mW[4]
OP293, OP298	200 mW[4]
Power Dissipation, Full Heat Sink	
OP290, OP291, OP292, OP295, OP296, OP297	1.11 W[5]
OP293, OP298	400 mW[5]
Power Dissipation	
OP294, OP299	180 mW

Notes:
1. For OP290, OP291, OP292, OP295, OP296 and OP297, refer to graph of Maximum Peak Pulse Current vs Pulse Width.
2. For all OPs in this series, RMA flux is recommended. Duration can be extended to 10 second maximum when soldering. A maximum of 20 grams force may be applied to the leads when flow soldering.
3. For OP290, OP291, OP292, OP295, OP296 and OP297, measured in free-air. Derate linearly 3.33 mW/° C above 25° C.
4. For OP290, OP291and OP292, mounted on 1/16" (1.6 mm) thick PCBoard with each lead soldered through 80 mil square lands 0.250" (6.35 mm) below flange of device. Derate linearly 5.33 mW/°C above 62.5°. For OP293 and OP298, mounted on 1/16" (1.60 mm) thick PCBoard with each lead soldered through 80 mil square lands 0.250" (6.35 mm) below flange of device. Derate power dissipation linearly 2.00 mW/°C above 25° C (normal use). For OP295, OP296 and OP297, mounted on 1/16" (1.6 mm) thick PCBoard with each lead soldered through 80 mil square lands 0.250" (6.35 mm) below flange of device. Derate linearly 5.33 mW/°C above 25° C.
5. Immersed in silicone fluid to simulate infinite heat sink. For OP290, OP291 and OP292, derate linearly 11.1 mW/°C above 95°C. For OP293 and OP298, derate power dissipation linearly 2.50 mW/° C above 25° C. For OP295, OP296 and OP297, derate linearly 11.1 mW/° C above 25° C.

Electrical Characteristics (T_A = 25°C unless otherwise noted)

SYMBOL	PARAMETER	MIN	TYP	MAX	UNITS	TEST CONDITIONS
Input Diode						
$E_{E\,(APT)}$[2]	Apertured Radiant Incidence OP290A OP290B OP290C	 210 180 150	 - - -	 - 300 -	mW/cm²	I_F = 1.50 A[1][2] Measured into a 0.250" [6.35mm] aperture 0.2" (5.08 mm) from the tip of the lens.
	OP291A OP291B OP291C	16 13 10	- - -	- 26 -		I_F = 100 mA[1][2] Measured into a 0.250" [6.35mm] aperture 0.2" (5.08 mm) from the tip of the lens.
	OP292A OP292B OP292C	2.7 2.2 1.7	- 3.6 -	- 4.4 -		I_F = 20 mA[1][2] Measured into a 0.250" [6.35mm] aperture 0.2" (5.08 mm) from the tip of the lens.
	OP293A OP293B OP293C	16 13 10	- 22 -	- 26 -		I_F = 100 mA[1][2] Measured into a 0.250" [6.35mm] aperture 0.2" (5.08 mm) from the tip of the lens.
	OP294	0.50	-	1.50		I_F = 5 mA[1][2] Measured into a 0.250" [6.35mm] aperture 0.200" (5.08mm) from the tip of the lens.
	OP295A OP295B OP295C	44 33 22	- - -	- 77 -		I_F = 1.50 A[1][2] Measured into a 0.250" [6.35mm] aperture 1.129" (28.7 mm) from the tip of the lens.
	OP296A OP296B OP296C	3.6 2.6 1.6	- - -	- 6.6 -		I_F = 100 mA[1][2] Measured into a 0.250" [6.35mm] aperture 1.129" (28.7 mm) from the tip of the lens.
	OP297FAB OP297A OP297B OP297C	2.4 0.7 0.5 0.3	- - 1.0 -	- - 1.3 -		I_F = 20 mA[1][2] Measured into a 0.250" [6.35mm] aperture 1.129" (28.7 mm) from the tip of the lens.
	OP298A OP298B OP298C	3.0 2.4 1.8	- - -	- 4.8 -		I_F = 100 mA[1][2] Measured into a 0.250" [6.35mm] aperture 0.2" (5.08 mm) from the tip of the lens.
	OP298AA OP298AB OP298AC OP298AD	3.5 3.5 6.5 8.5	- - - -	- 8.5 11.5 -		I_F = 100 mA[1][2] Measured into a 0.250" [6.35mm] aperture 1.129" (28.7 mm) from the tip of the lens.
	OP299	0.15	-	0.45		I_F = 100 mA[1][2] Measured into a 0.250" [6.35mm] aperture 1.129" (28.7 mm) from the tip of the lens.

Notes:
1. Measurement is taken at the end of a single 100 µs pulse. Heating due to increased pulse rate or pulse width will cause a decrease in reading.
2. Measurement of the average apertured radiant energy incident upon a sensing area 0.250" (6.35 mm) in diameter perpendicular to and centered on the mechanical axis of the lens and the specified distance from the end of the device. On all models in this series, $E_{E(APT)}$ is not necessarily uniform within the measured area.
3. Measurement is taken at the end of a single 10 ms pulse. Heating due to increased pulse rate or pulse width will cause a decrease in reading.

OPTEK Technology Inc. — 1645 Wallace Drive, Carrollton, Texas 75006
Phone: (972) 323-2200 or (800) 341-4747 FAX: (972) 323-2396 sensors@optekinc.com www.optekinc.com

Electrical Characteristics (T_A = 25°C unless otherwise noted)

SYMBOL	PARAMETER	MIN	TYP	MAX	UNITS	TEST CONDITIONS
Input Diode						
V_F	Forward Voltage[3]					
	OP290, OP295	-	-	4.00	V	I_F = 1.50 A
	OP291, OP296	-	-	2.00		I_F = 100 mA
	OP292, OP297, OP297FAB	-	-	1.75		I_F = 20 mA
	OP293, OP298 (A, B, C)	-	-	2.00		I_F = 1.50 A
	OP298 (AA, AB, AC, AD)	-	-	2.00		I_F = 100 mA
	OP294, OP299	-	-	1.50		I_F = 5 mA
I_R	Reverse Current[3]					
	OP290, OP292	-	-	10	µA	V_R= 5 V
	OP291, OP293, OP298 (A, B, C), OP296	-	-	100		V_R= 2 V
	OP298 (AA, AB, AC, AD)	-	-	100		V_R= 2 V
	OP294, OP299	-	-	10		V_R= 2 V
	OP295, OP297	-	-	10		V_R= 5 V
	OP297FAB	-	-	15		V_R= 5 V
λ_P	Wavelength at Peak Emission					
	OP290, OP291, OP292, OP293, OP294, OP295, OP296, OP297, OP298 (A, B, C), OP299	-	890	-	nm	I_F = 10 mA
	OP297FAB, OP298 (AA, AB, AC, AD)	-	875	-		
B	Spectral Bandwidth between Half Power Points	-	80	-	nm	I_F = 10 mA
$\Delta\lambda_P/\Delta T$	Spectral Shift with Temperature	-	+0.18	-	nm/°C	I_F = Constant
θ_{HP}	Emission Angle at Half Power Points					
	OP290, OP291, OP292, OP294	-	50	-	Degree	I_F = 20 mA
	OP293	-	60	-		
	OP295, OP296, OP297, OP299	-	20	-		
	OP298	-	25	-		
t_r	Output Rise Time	-	500	-	ns	$I_{F(PK)}$=100 mA, PW=10 µs, and D.C.=10.0%
t_f	Output Fall Time	-	250	-	ns	

High Power Infrared Emitting Diode, 940 nm, GaAlAs/GaAs

94 8389

FEATURES

- Package type: leaded
- Package form: T-1¾
- Dimensions (in mm): ∅ 5
- Peak wavelength: λ_p = 940 nm
- High reliability
- High radiant power
- High radiant intensity
- Angle of half intensity: φ = ± 30°
- Low forward voltage
- Suitable for high pulse current operation
- Good spectral matching with Si photodetectors
- Compliant to RoHS directive 2002/95/EC and in accordance to WEEE 2002/96/EC
- Halogen-free according to IEC 61249-2-21 definition

RoHS COMPLIANT HALOGEN FREE

DESCRIPTION

TSAL7600 is an infrared, 940 nm emitting diode in GaAlAs/GaAs technology with high radiant power molded in a clear, untinted plastic package.

APPLICATIONS

- Infrared remote control units with high power requirements
- Free air transmission systems
- Infrared source for optical counters and card readers

PRODUCT SUMMARY

COMPONENT	I_e (mW/sr)	φ (deg)	λ_p (nm)	t_r (ns)
TSAL7600	25	± 30	940	800

Note

Test conditions see table "Basic Characteristics"

ORDERING INFORMATION

ORDERING CODE	PACKAGING	REMARKS	PACKAGE FORM
TSAL7600	Bulk	MOQ: 4000 pcs, 4000 pcs/bulk	T-1¾

Note

MOQ: minimum order quantity

ABSOLUTE MAXIMUM RATINGS

PARAMETER	TEST CONDITION	SYMBOL	VALUE	UNIT
Reverse voltage		V_R	5	V
Forward current		I_F	100	mA
Peak forward current	t_p/T = 0.5, t_p = 100 µs	I_{FM}	200	mA
Surge forward current	t_p = 100 µs	I_{FSM}	1.5	A
Power dissipation		P_V	160	mW
Junction temperature		T_j	100	°C
Operating temperature range		T_{amb}	- 40 to + 85	°C
Storage temperature range		T_{stg}	- 40 to + 100	°C
Soldering temperature	t ≤ 5 s, 2 mm from case	T_{sd}	260	°C
Thermal resistance junction/ambient	J-STD-051, leads 7 mm soldered on PCB	R_{thJA}	230	K/W

Note

T_{amb} = 25 °C, unless otherwise specified

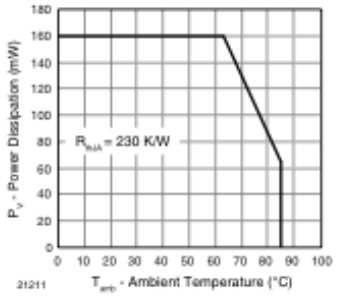


Fig. 1 - Power Dissipation Limit vs. Ambient Temperature

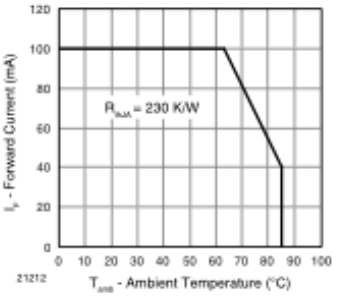


Fig. 2 - Forward Current Limit vs. Ambient Temperature

BASIC CHARACTERISTICS						
PARAMETER	**TEST CONDITION**	**SYMBOL**	**MIN.**	**TYP.**	**MAX.**	**UNIT**
Forward voltage	I_F = 100 mA, t_p = 20 ms	V_F		1.35	1.6	V
	I_F = 1 A, t_p = 100 µs	V_F		2.6	3	V
Temperature coefficient of V_F	I_F = 1 mA	TK_{VF}		- 1.8		mV/K
Reverse current	V_R = 5 V	I_R			10	µA
Junction capacitance	V_R = 0 V, f = 1 MHz, E = 0	C_j		25		pF
Radiant intensity	I_F = 100 mA, t_p = 20 ms	I_e	15	25	75	mW/sr
	I_F = 1 A, t_p = 100 µs	I_e	120	200		mW/sr
Radiant power	I_F = 100 mA, t_p = 20 ms	ϕ_e		35		mW
Temperature coefficient of ϕ_e	I_F = 20 mA	$TK\phi_e$		- 0.6		%/K
Angle of half intensity		φ		± 30		deg
Peak wavelength	I_F = 100 mA	λ_p		940		nm
Spectral bandwidth	I_F = 100 mA	$\Delta\lambda$		50		nm
Temperature coefficient of λ_p	I_F = 100 mA	$TK\lambda_p$		0.2		nm/K
Rise time	I_F = 100 mA	t_r		800		ns
Fall time	I_F = 100 mA	t_f		800		ns
Virtual source diameter		d		1.8		mm

Note

T_{amb} = 25 °C, unless otherwise specified

Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32K Bytes of In-System Self-Programmable Flash progam memory (ATmega48PA/88PA/168PA/328P)
  - 256/512/512/1K Bytes EEPROM (ATmega48PA/88PA/168PA/328P)
  - 512/1K/1K/2K Bytes Internal SRAM (ATmega48PA/88PA/168PA/328P)
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I^2C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V for ATmega48PA/88PA/168PA/328P
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 20 MHz @ 1.8 - 5.5V
- Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48PA/88PA/168PA/328P:
  - Active Mode: 0.2 mA
  - Power-down Mode: 0.1 µA
  - Power-save Mode: 0.75 µA (Including 32 kHz RTC)

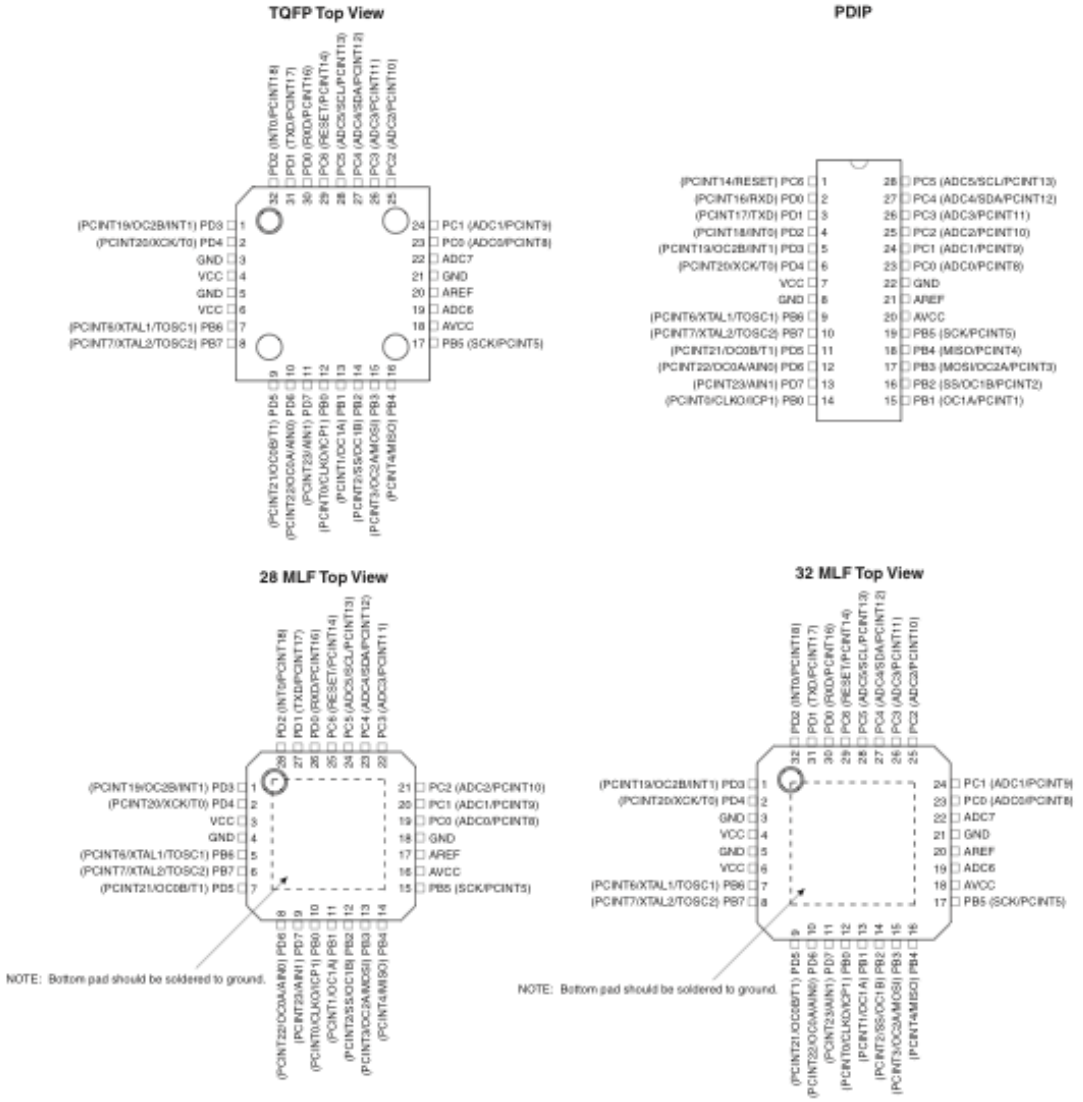
8-bit AVR® Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash

ATmega48PA
ATmega88PA
ATmega168PA
ATmega328P

Rev. 8161D–AVR–10/09

1. Pin Configurations

Figure 1-1. Pinout ATmega48PA/88PA/168PA/328P



Features

- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory segments
  - 4/8/16 Kbytes of In-System Self-programmable Flash program memory
  - 256/512/512 Bytes EEPROM
  - 512/1K/1K Bytes Internal SRAM
  - Write/Erase cyles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C[1]
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
  - 6-channel 10-bit ADC in PDIP Package
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I^2C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - DebugWIRE On-Chip Debug System
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Five Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, and Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8V - 5.5V for ATmega48V/88V/168V
  - 2.7V - 5.5V for ATmega48/88/168
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - ATmega48V/88V/168V: 0 - 4 MHz @ 1.8V - 5.5V, 0 - 10 MHz @ 2.7V - 5.5V
  - ATmega48/88/168: 0 - 10 MHz @ 2.7V - 5.5V, 0 - 20 MHz @ 4.5V - 5.5V
- Low Power Consumption
  - Active Mode:
    - 250 µA at 1 MHz, 1.8V
    - 15 µA at 32 kHz, 1.8V (including Oscillator)
  - Power-down Mode:
    - 0.1 µA at 1.8V

Note: 1. See "Data Retention" on page 7 for details.

8-bit AVR® Microcontroller with 4/8/16K Bytes In-System Programmable Flash

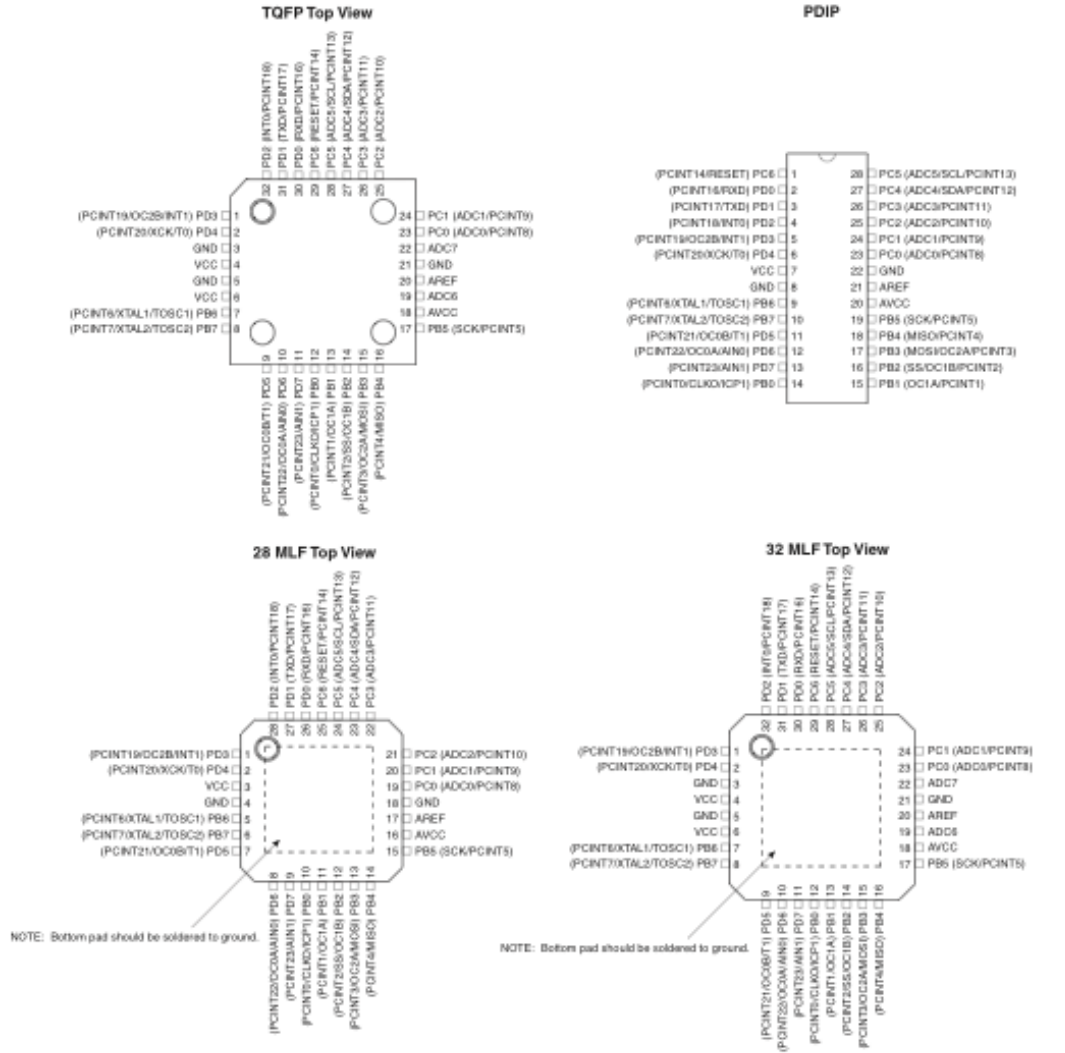
ATmega48/V
ATmega88/V
ATmega168/V

Summary

Rev. 2545SS–AVR–07/10

1. Pin Configurations

Figure 1-1. Pinout ATmega48/88/1682545SS

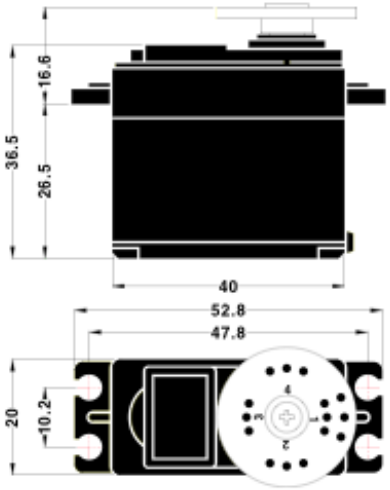


PREPARED BY JUN HEE, LEE
UPDATE:APR 01, 2002

ANNOUNCED SPECIFICATION OF HS-311 STANDARD SERVO

1.TECHNICAL VALUE

CONTROL SYSTEM	:+PULSE WIDTH CONTROL 1500usec NEUTRAL	
OPERATING VOLTAGE RANGE	:4.8V TO 6.0V	
TEST VOLTAGE	:AT 4.8V	AT 6.0V
OPERATING SPEED	:0.19sec/60 AT NO LOAD	0.15sec/60 AT NO LOAD
STALL TORQUE	:3.0kg.cm(42oz.in)	3.5kg.cm(48.60oz.in)
IDLE CURRENT	:7.4mA AT STOPPED	7.7mA AT STOPPED
RUNNING CURRENT	:160mA/60 AT NO LOAD	180mA/60 AT NO LOAD
STALL CURRENT	:700mA	800mA
DEAD BAND WIDTH	:5usec	5usec
OPERATING TRAVEL	:40 /ONE SIDE PULSE TRAVELING 400usec	
DIRECTION	:CLOCK WISE/PULSE TRAVELING 1500 TO 1900usec	
MOTOR TYPE	:CORED METAL BRUSH	
POTENTIOMETER TYPE	:4 SLIDER/DIRECT DRIVE	
AMPLIFIER TYPE	:ANALOG CONTROLLER & TRANSISTOR DRIVER	
DIMENSIONS	:40x20x36.5mm(1.57x0.78x1.43in)	
WEIGHT	:43g(1.51oz)	
BALL BEARING	:TOP/RESIN BUSHING	
GEAR MATERIAL	:RESIN	
HORN GEAR SPLINE	:24 SEGMENTS/ 5.76	
SPLINED HORNS	:SUPER/R-XA	
CONNECTOR WIRE LENGTH	:300mm(11.81in)	
CONNECTOR WIRE STRAND COUNTER	:40EA	
CONNECTOR WIRE GAUGE		



2.FEATURES

LONG LIFE POTENTIOMETER, TOP RESIN BUSHING

3.APPLICATIONS

AIRCRAFT 20-40 SIZE,STEERING AND THROTTLE SERVO FOR CARS, TRUCK AND BOATS

4.ACCESSORY & OPTION

CASE SET/	GEAR SET/	HORN SET/
HS322T:1EA	HS322G1:1EA	R-XA:1EA
HS322M:1EA	HS322G2:1EA	
HS322L:1EA	HS322G3:1EA	
PH/T-2 2x30 NI:4EA	HS322G4:1EA	
	HS300RB:1EA	

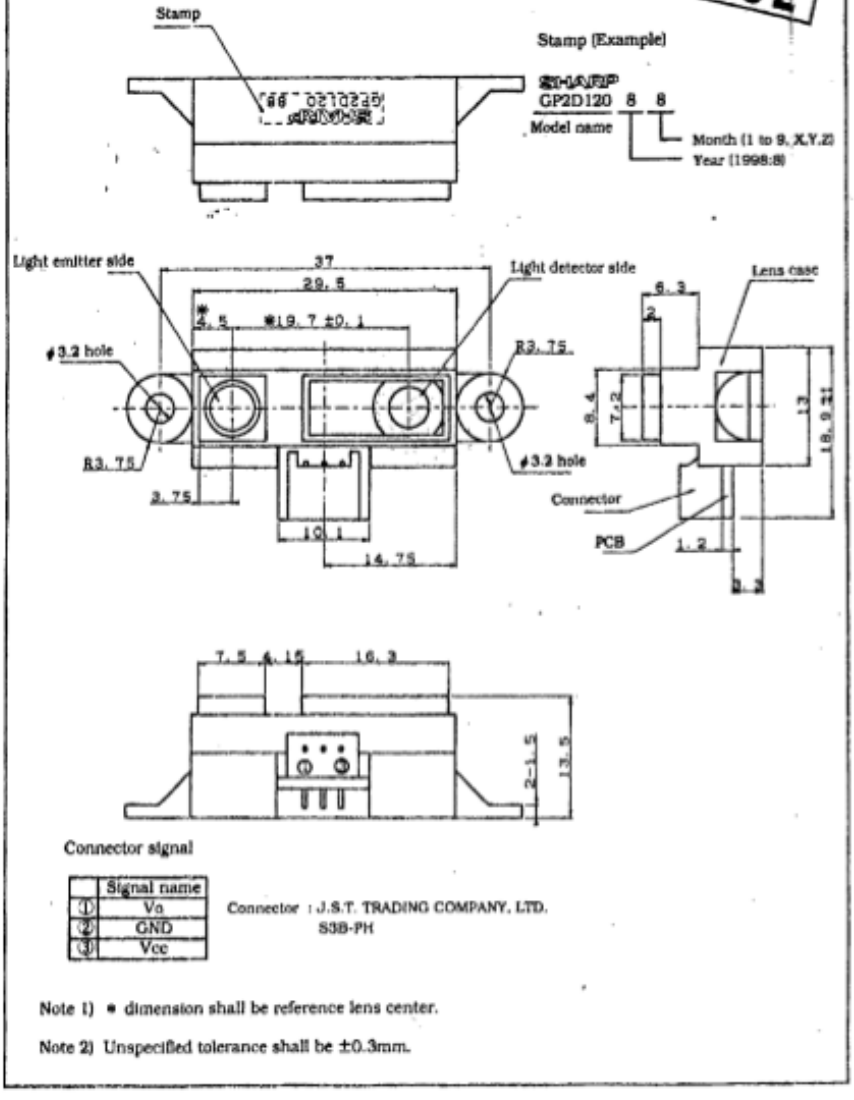
HITEC RCD KOREA INC.

SHARP CORPORATION

ED-99170 | August 30, 1999
PAGE 3/9

REFERENCE

2. Outline (Drawing No. SOD03596) Scale : 2/1 Unit : mm



Connector signal

	Signal name
①	Vo
②	GND
③	Vcc

Connector : J.S.T. TRADING COMPANY, LTD.
S3B-PH

Note 1) * dimension shall be reference lens center.

Note 2) Unspecified tolerance shall be ±0.3mm.

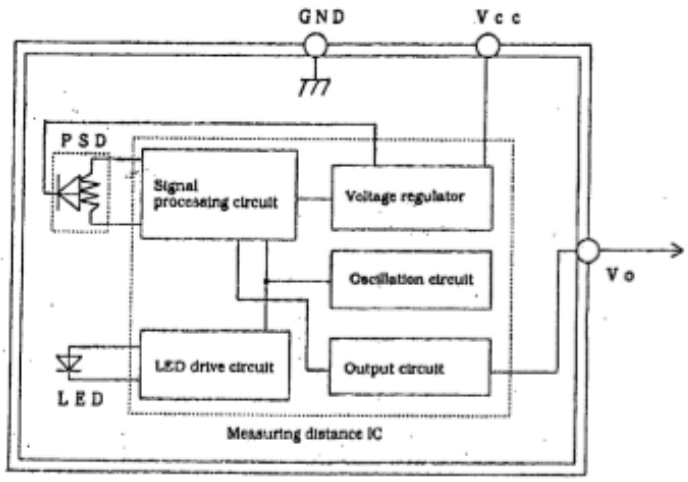
SHARP CORPORATION

ED-99170	August 30, 1999
SHARP No. GP2D120	PAGE 4/9

REFERENCE

3. Ratings and characteristics

3-1 Constitution diagram



3-2 Absolute maximum ratings

(Ta=25℃, Vcc=5V)

Parameter	Symbol	Rating	Unit	Remark
Supply voltage	Vcc	-0.3 to +7	V	
Output terminal voltage	Vo	-0.3 to Vcc+0.3	V	
Operating temperature	Topr	-10 to +60	℃	
Storage temperature	Tstg	-40 to +70	℃	

・Operating Supply Voltage

Parameter	Symbol	Rating	Unit	Remark
Operating Supply Voltage	Vcc	4.5 to 5.5	V	

SHARP CORPORATION

ED-99170 | August 30, 1999
MODEL No. GP2D120 | PAGE 5/9

REFERENCE

3-3 Electro-optical Characteristics

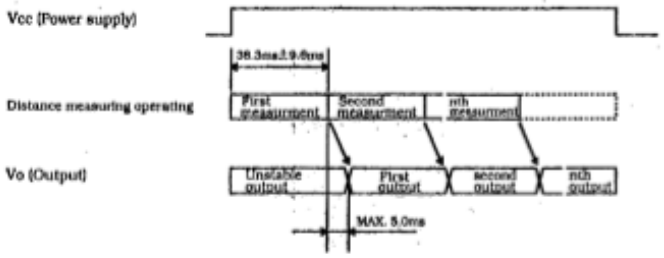
(Ta=25°C, Vcc=5V)

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Measuring distance range	ΔL	(*1)	4	-	30	cm
Output terminal voltage	Vo	L=30cm (*1)	0.25	0.4	0.55	V
Output voltage difference	ΔVo	Output change at L change (30cm → 4cm) (*1)	1.95	2.25	2.55	V
Average supply current	Icc	L=30cm (*1)	-	33	50	mA

■ L : Distance to reflective object

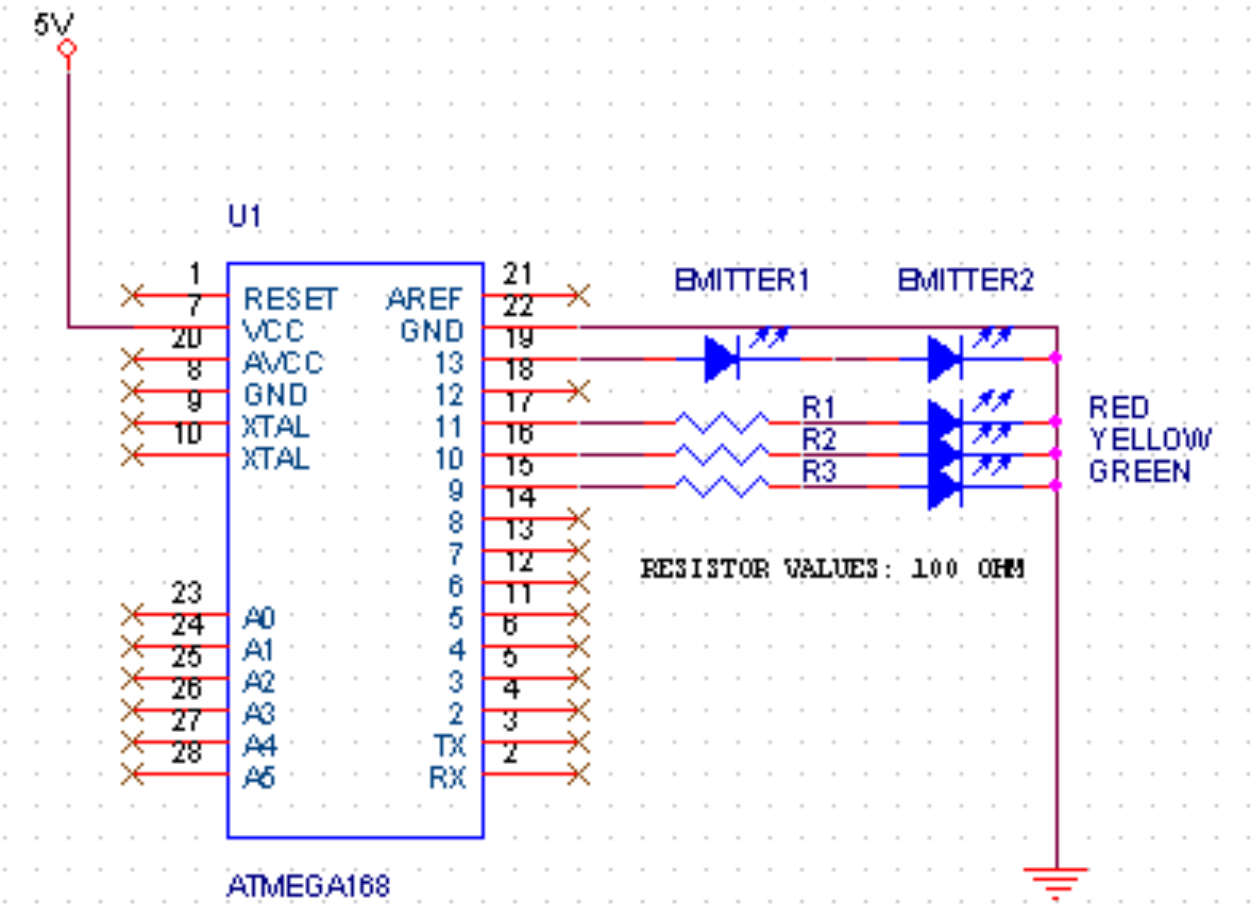
(*1) Using reflective object : White paper (Made by Kodak Co. Ltd. gray cards R-27 • white face, reflective ratio : 90%)

3-4 Timing chart

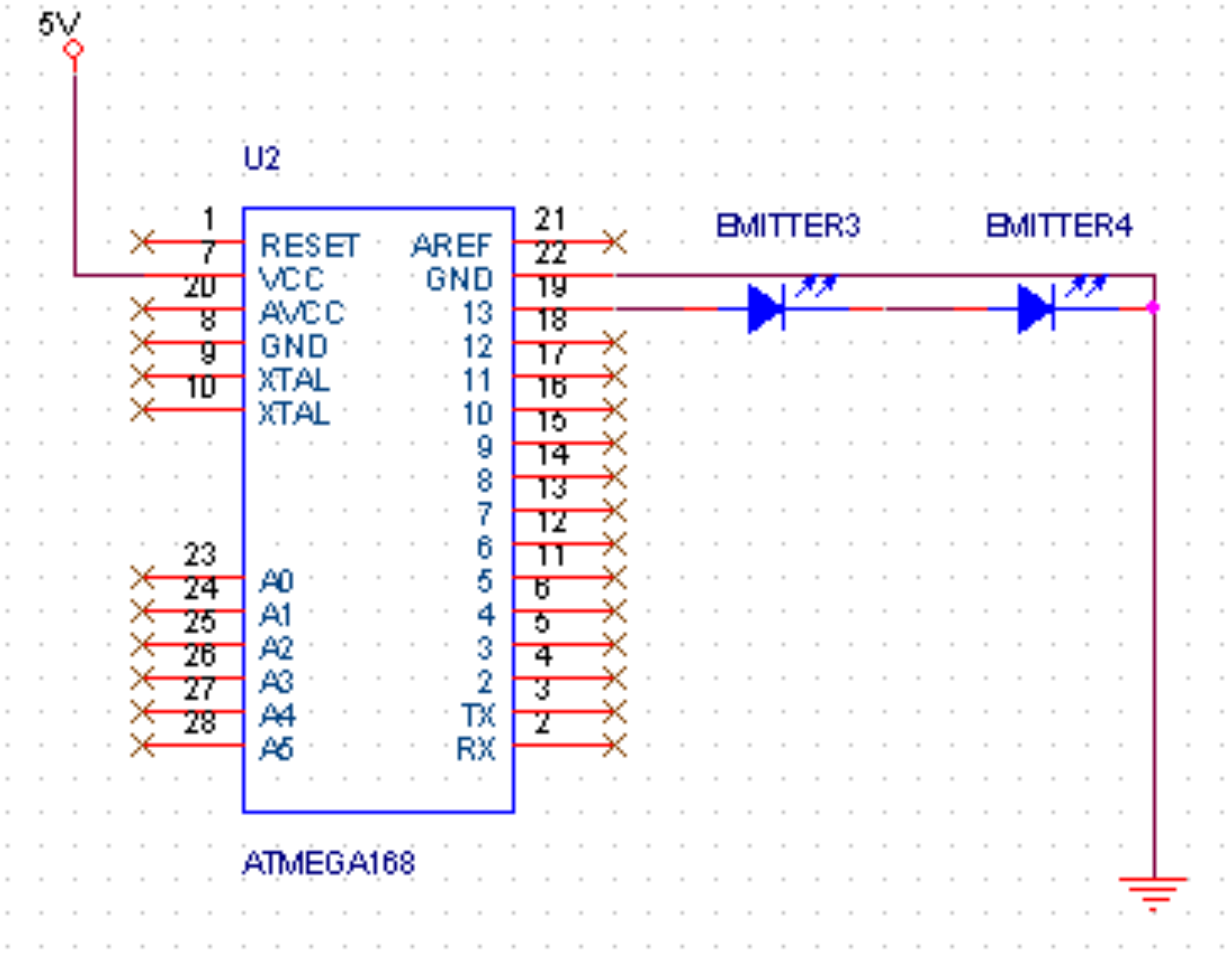


APPENDIX B: SCHEMATICS

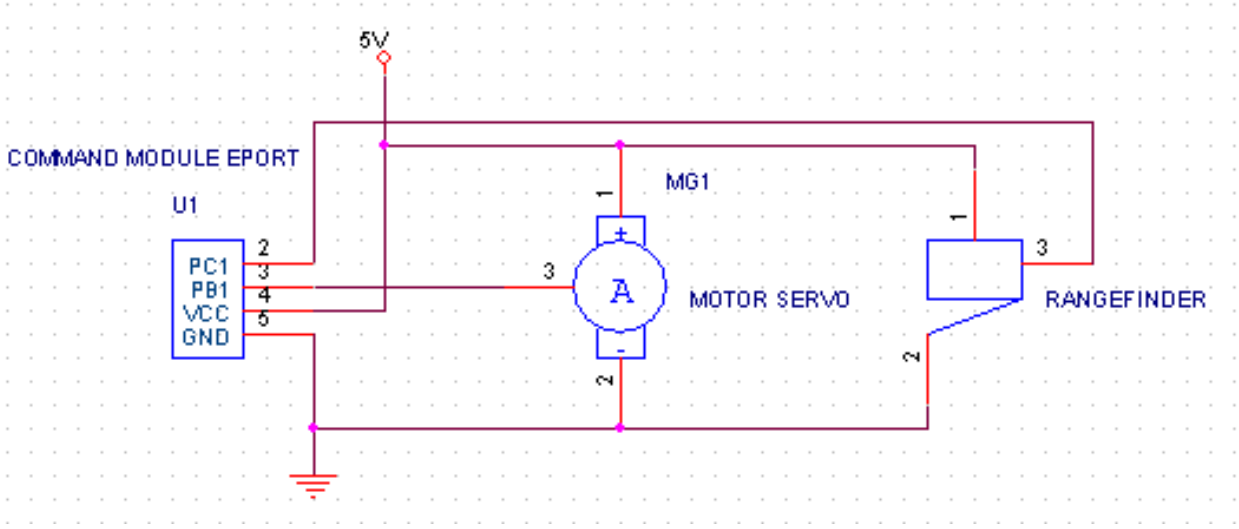
Arduino Traffic Light



Arduino Speed Limit Sign



Command Module with Servo & Rangefinder



APPENDIX C: SOURCE CODE

MAIN PROGRAM

```
/* SGRVP.c
 * Designed to run on Create Command Module
 *
 * The basic architecture of this program can be re-used to easily
 * write a wide variety of Create control programs.  All sensor values
 * are polled in the background (using the serial rx interrupt) and
 * stored in the sensors array as long as the function
 * delayAndUpdateSensors() is called periodically.  Users can send commands
 * directly a byte at a time using byteTx() or they can use the
 * provided functions, such as baud() and drive().
 */
/*****************************************************************************
*       This program uses an IR Rangefinder (rotated by a servo) to scan each
*       cell on a map for obstacles. If found, it will avoid them and find a free
*       cell to move into. It also continuously listens for IR (stoplight signal)
*       interrupts. If interrupted, it will pause for a specified amount of time
*       and wait for the light to change to green. Program ends when the Goal
*       position is found.
*
*       April 13th, 2007
*
*****************************************************************************/
```

```
// Includes
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/eeprom.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "oi.h"

//AVRlib includes
#include "global.h"            // include global settings
#include "a2d.h"               // include A/D converter function library
#include "wave_front.c"        //include Wavefront Algorithm

// Constants
#define RESET_SONG 0
#define START_SONG 1
#define BUMP_SONG  2
#define END_SONG   3

//define port functions; example: PORT_ON( PORTD, 6);
#define PORT_ON( port_letter, number )              port_letter |= (1<<number)
#define PORT_OFF( port_letter, number )             port_letter &= ~(1<<number)
#define PORT_ALL_ON( port_letter, number )          port_letter |= (number)
```

```
#define PORT_ALL_OFF( port_letter, number )          port_letter &= ~(number)
#define FLIP_PORT( port_letter, number )          port_letter ^= (1<<number)
#define PORT_IS_ON( port_letter, number )          ( port_letter & (1<<number) )
#define PORT_IS_OFF( port_letter, number )          !( port_letter & (1<<number) )

// Global variables
volatile uint16_t timer_cnt = 0;
volatile uint8_t timer_on = 0;
volatile uint8_t sensors_flag = 0;
volatile uint8_t sensors_index = 0;
volatile uint8_t sensors_in[Sen6Size];
volatile uint8_t sensors[Sen6Size];
volatile int16_t distance = 0;
signed long int angle = 0;
volatile uint16_t update_delay = 16;//minimum of 15ms per sensor update allowed!!!
volatile uint8_t range = 0;
int j=0;

// Scanning Variables
unsigned int scan_thresh=0;//threshold value of Sharp IR for scanner
unsigned long int scan_angle=750;//position of scanner, in units of servo command
unsigned int tmp=255;
unsigned int object_found=0;
```

```
int old_state=1;//records current pointed direction of robot: 1 is up, 2 is right, 3 is down, and 4 is left (clockwise)

int new_state=1;//required direction of robot


// Functions

void byteTx(uint8_t value);

void delayMs(uint16_t time_ms);

void delayAndUpdateSensors(unsigned int time_ms);

void initialize(void);

void powerOnRobot(void);

void baud(uint8_t baud_code);

void drive(int16_t velocity, int16_t radius);

void drive_distance(int16_t velocity, int16_t radius, int16_t distance_togo);

void drive_angle(int16_t velocity, int16_t radius, int16_t angle_togo);

void defineSongs(void);


//Other Functions

void rotate_CCW(signed long int angle_tmp, signed long int velocity);

void rotate_CW(signed long int angle_tmp, signed long int velocity);

void straight(signed long int distance_tmp, signed long int velocity);

void stop(void);

void scan(unsigned int scan_thresh);

void servo_scan(unsigned long int speed);

void delay_cycles(unsigned long int cycles);
```

```
void find_walls(void);
void CheckIRRemote(void);


int main (void)
{
        if (j>0) // counts interrupts and directs to pick up where robot left off in map
        {
                goto PICKUP;
        }
 uint8_t leds_cnt = 99;
 uint8_t leds_on = 1;

// Set up Create and module
 initialize();
 LEDBothOff;
 powerOnRobot();
 byteTx(CmdStart);
 baud(Baud28800);
 defineSongs();
 byteTx(CmdControl);
 byteTx(CmdFull);

 // Stop just as a precaution
 drive(0, RadStraight);
```

```
 // Play the reset song and wait while it plays
 byteTx(CmdPlay);
 byteTx(RESET_SONG);
 delayAndUpdateSensors(750);

 while(timer_on)
 {
 CheckIRRemote();
 }

for(;;)
 {
        delayAndUpdateSensors(50);

if(++leds_cnt >= 100)
  {
    leds_cnt = 0;
    leds_on = !leds_on;

if(leds_on)
    {
      byteTx(CmdLeds);
      byteTx(LEDsBoth);
      byteTx(128);
```

```
      byteTx(255);
      LEDBothOff;
    }
    else
    {
      byteTx(CmdLeds);
      byteTx(0x00);
      byteTx(0);
      byteTx(0);
      LEDBothOn;
    }
  }
delayAndUpdateSensors(10);

if(UserButtonPressed)
  {
    // Play start song and wait
    byteTx(CmdPlay);
    byteTx(START_SONG);
    delayAndUpdateSensors(2500);

//reset encoder function
    angle = 0;
          distance = 0;
```

```
        //store robot location in map
        map[robot_x][robot_y]=robot;
        //store robot location in map
        map[goal_x][goal_y]=goal;

    // Drive around until a button or unsafe condition is detected
        while(!(UserButtonPressed)
        && (!sensors[SenCliffL])
        && (!sensors[SenCliffFL])
        && (!sensors[SenCliffFR])
        && (!sensors[SenCliffR])
        && (!sensors[SenChAvailable]) && map[robot_x][robot_y]!=goal)//program ends when robot reaches goal

    {
                find_walls();
                new_state=propagate_wavefront(robot_x,robot_y,goal_x,goal_y);

                //////////////wavefront code//////////////

PICKUP: // goes back here after interrupt
                while(new_state!=old_state && new_state!=0)
                        {
                        //if not pointed in the right direction, rotate
                        if (abs(old_state - new_state) == 2)//rotate 180 degrees
```

```
                    rotate_CCW(180,250);

                else if ((signed int)(old_state - new_state) == -1 || (signed
int)(old_state - new_state) == 3)//rotate 90 degrees CW

                    rotate_CW(90,150);

                else if ((signed int)(old_state - new_state) == 1 || (signed int)(old_state
- new_state) == -3)//rotate 90 degrees CCW

                    rotate_CCW(90,150);


                //make new state the old state

                old_state=new_state;


                //recheck with new location

                find_walls();


                //find new location to go to

                new_state=propagate_wavefront(robot_x,robot_y,goal_x,goal_y);

                }


        //there is no solution

        if (new_state==0)

                {

                byteTx(CmdPlay);//play error song

                byteTx(BUMP_SONG);

        delayMs(2500);

                //reset the entire map
```

```
            clear_map();
            }

        //go to new location
        else //move if there is a solution, otherwise dont move forward
            {
            straight(cell_size,100);//distance, velocity

            //update new location of robot
            if (new_state==1)
                robot_x--;
            if (new_state==2)
                robot_y++;
            if (new_state==3)
                robot_x++;
            if (new_state==4)
                robot_y--;
            }

        //make new state the old state
        if(new_state!=0)
            old_state=new_state;

// wait a little more than one robot tick for sensors to update
delayAndUpdateSensors(update_delay);
```

```
    }

    // Stop driving
    drive(0, RadStraight);

    // Play end song and wait
    delayAndUpdateSensors(500);

    byteTx(CmdPlay);
    byteTx(END_SONG);
    delayAndUpdateSensors(2000);

   }
  }
}

void CheckIRRemote(void) //function for interfacing with “traffic light”
{
        uint8_t IRchar;
        if(sensors[SenIRChar]==0xFF)
                return;
        IRchar = sensors[SenIRChar];
        switch (IRchar)
        {
```

```
        case IRForward:
                        j++; //increment counter
                        main();//go back to main function

        case IRPause: //stops robot in its tracks
                        //drive_speed=0;
                 drive(0, RadStraight);
                        break;
                default:
                        break;
        }
        return;
}

// Serial receive interrupt to store sensor values
SIGNAL(SIG_USART_RECV)
{
  uint8_t temp;
temp = UDR0;
if(sensors_flag)
  {
    sensors_in[sensors_index++] = temp;
    if(sensors_index >= Sen6Size)
      sensors_flag = 0;
```

```
 }
}

// Timer 1 interrupt to time delays in ms
SIGNAL(SIG_OUTPUT_COMPARE1A)
{
 if(timer_cnt)
   timer_cnt--;
 else
   timer_on = 0;
}

// Transmit a byte over the serial port
void byteTx(uint8_t value)
{
 while(!(UCSR0A & _BV(UDRE0))) ;
 UDR0 = value;
}

// Delay for the specified time in ms without updating sensor values
void delayMs(uint16_t time_ms)
{
 timer_on = 1;
 timer_cnt = time_ms;
```

```
 while(timer_on);
 }

// Delay for the specified time in ms and update sensor values
//Create updates its sensors and replies to sensor requests at a rate of
//67 Hz, or every 15 milliseconds. Do not send sensor requests faster than this.
void delayAndUpdateSensors(uint16_t time_ms)
{
 uint8_t temp;
 timer_on = 1;
 timer_cnt = time_ms;
 while(timer_on)
 {
  if(!sensors_flag)
  {
    for(temp = 0; temp < Sen6Size; temp++)
      sensors[temp] = sensors_in[temp];

    // Update running totals of distance and angle
    distance += (int)((sensors[SenDist1] << 8) | sensors[SenDist0]);
    angle += (int)((sensors[SenAng1] << 8) | sensors[SenAng0]);
          byteTx(CmdSensors);
    byteTx(6);
    sensors_index = 0;
```

```
    sensors_flag = 1;
   }
  }
  range=a2dConvert8bit(1);
  CheckIRRemote();
  return;
}

// Initialize the Command Module's ATmega168 microcontroller
void initialize(void)
{
  cli();

  // Set I/O pins
  DDRB = 0x12;          //12, 10   //0b00010010 (4 must be output, 5 must be input)// user a
1 for servos
  PORTB = 0xCF; //CD, CF   //0b11001111
  DDRC = 0x00;  //configure all C ports for input
  PORTC = 0x00; //make sure pull-up resistors are turned off
  DDRD = 0xE6;  //0b11100110 (1's for input)
  PORTD = 0x7D; //0b01111101 (1's for output)

  // Set up timer 1 to generate an interrupt every 1 ms
  TCCR1A = 0x00;
  TCCR1B = (_BV(WGM12) | _BV(CS12));
```

```
  OCR1A = 71;
  TIMSK1 = _BV(OCIE1A);

  // Set up the serial port with rx interrupt
  UBRR0 = 19;
  UCSR0B = (_BV(RXCIE0) | _BV(TXEN0) | _BV(RXEN0));
  UCSR0C = (_BV(UCSZ00) | _BV(UCSZ01));

  /* initialize I2C (TWI) clock: 100 kHz clock, TWPS = 0 => prescaler = 1 */
  TWSR = 0;
  TWBR = (F_CPU / 100000UL - 16) / 2;

  // Set up the ADC on pin C5
  a2dInit(); // initialize analog to digital converter (ADC)
  a2dSetPrescaler(ADC_PRESCALE_DIV32); // configure ADC scaling
  a2dSetReference(ADC_REFERENCE_AVCC); // configure ADC reference voltage

  // Turn on interrupts
  sei();
}

void powerOnRobot(void)
{
  // If Create's power is off, turn it on
  if(!RobotIsOn)
  {
```

```
    while(!RobotIsOn)
    {
       RobotPwrToggleLow;
       delayMs(500);  // Delay in this state
       RobotPwrToggleHigh;  // Low to high transition to toggle power
       delayMs(100);  // Delay in this state
       RobotPwrToggleLow;
    }
    delayMs(3000);  // Delay for startup
  }
}

// Switch the baud rate on both Create and module
void baud(uint8_t baud_code)
{
  if(baud_code <= 11)
  {
    byteTx(CmdBaud);
    UCSR0A |= _BV(TXC0);
    byteTx(baud_code);
    // Wait until transmit is complete
    while(!(UCSR0A & _BV(TXC0))) ;
```

```
cli();

// Switch the baud rate register
if(baud_code == Baud115200)
  UBRR0 = Ubrr115200;
else if(baud_code == Baud57600)
  UBRR0 = Ubrr57600;
else if(baud_code == Baud38400)
  UBRR0 = Ubrr38400;
else if(baud_code == Baud28800)
  UBRR0 = Ubrr28800;
else if(baud_code == Baud19200)
  UBRR0 = Ubrr19200;
else if(baud_code == Baud14400)
  UBRR0 = Ubrr14400;
else if(baud_code == Baud9600)
  UBRR0 = Ubrr9600;
else if(baud_code == Baud4800)
  UBRR0 = Ubrr4800;
else if(baud_code == Baud2400)
  UBRR0 = Ubrr2400;
else if(baud_code == Baud1200)
  UBRR0 = Ubrr1200;
else if(baud_code == Baud600)
```

```
    UBRR0 = Ubrr600;
  else if(baud_code == Baud300)
    UBRR0 = Ubrr300;

  sei();

  delayMs(100);
 }
}

// Send Create drive commands in terms of velocity and radius
void drive(int16_t velocity, int16_t radius)
{
 byteTx(CmdDrive);
 byteTx((uint8_t)((velocity >> 8) & 0x00FF));
 byteTx((uint8_t)(velocity & 0x00FF));
 byteTx((uint8_t)((radius >> 8) & 0x00FF));
 byteTx((uint8_t)(radius & 0x00FF));
}

// Define songs to be played later
void defineSongs(void)
{
 // Reset song
 byteTx(CmdSong);
```

```
byteTx(RESET_SONG);
byteTx(4);
byteTx(60);
byteTx(6);
byteTx(72);
byteTx(6);
byteTx(84);
byteTx(6);
byteTx(96);
byteTx(6);

// Start song
byteTx(CmdSong);
byteTx(START_SONG);
byteTx(6);
byteTx(69);
byteTx(18);
byteTx(72);
byteTx(12);
byteTx(74);
byteTx(12);
byteTx(72);
byteTx(12);
```

```
byteTx(69);
byteTx(12);
byteTx(77);
byteTx(24);

// Bump song
byteTx(CmdSong);
byteTx(BUMP_SONG);
byteTx(2);
byteTx(74);
byteTx(12);
byteTx(59);
byteTx(24);

// End song
byteTx(CmdSong);
byteTx(END_SONG);
byteTx(5);
byteTx(77);
byteTx(18);
byteTx(74);
byteTx(12);
byteTx(72);
```

```
 byteTx(12);
 byteTx(69);
 byteTx(12);
 byteTx(65);
 byteTx(24);
}

//rotate a certain angle - 100 = 90 degrees!
void rotate_CCW(signed long int angle_tmp, signed long int velocity)
        {
        //Counter-clockwise angles are positive and clockwise angles are negative.
        drive(velocity, RadCCW);
        while(angle<angle_tmp)
                delayAndUpdateSensors(update_delay);//flaw is that if it goes above in the
middle of function, it will drift
        stop();
        delayAndUpdateSensors(update_delay);

        angle=angle-angle_tmp;//resets angle to 0; accounts for an over/undershoot
        angle=0;
        }
//rotate a certain angle - 100 = 90 degrees!
void rotate_CW(signed long int angle_tmp, signed long int velocity)
        {
```

```
	//Counter-clockwise angles are positive and clockwise angles are negative.

	drive(velocity, RadCW);

	while((-angle)<angle_tmp)

		delayAndUpdateSensors(update_delay);//flaw is that if it goes above in the middle of function, it will drift

	stop();

	delayAndUpdateSensors(update_delay);

	angle=angle-angle_tmp;//resets angle to 0; accounts for an over/undershoot

	angle=0;

	}

//drive straight function (use negative velocity to go in reverse)

void straight(signed long int distance_tmp, signed long int velocity)

	{

	drive(velocity, RadStraight);

	while(distance<distance_tmp)//flaw is that if it goes above in the middle of function, it will drift

		delayAndUpdateSensors(update_delay);

	stop();

	delayAndUpdateSensors(update_delay);


	distance=distance-distance_tmp;//resets angle to 0; accounts for an over/undershoot

	}
```

```
//stop motors
void stop(void)
	{
	drive(0, RadStraight);
	delayAndUpdateSensors(update_delay);
	}
//EDGE DETECTION
//this function causes scanning servo to center on edge of object
void scan(unsigned int scan_thresh)
	{
	//lower (-) goes right
	//higher (+) goes left

	/*psuedocode
	object is detected
		scan CW while object is detected (right)
	object not detected
		scan CCW until object detected (left)*/
	if (a2dConvert8bit(1) > scan_thresh)//object detected range
		{
		if (scan_angle>300) //overflow protection, servo cant rotate any further (minimum CCW, 500 is good)
			scan_angle-=10; //scan left, higher number makes it go faster
		}
```

```
	else //object not detected

		{

		if (scan_angle<=1200) //maximum position scanner can rotate to (maximum CW, 900 is good) 1200

			scan_angle+=10; //scan left

		}

	//servo scan code

	servo_scan(scan_angle);

	}


//UPDATE MAP BY SCANNING FOR WALLS

void find_walls(void)

	{

	//do not scan outside the border region

	if((old_state==1 && robot_x==0) || (old_state==2 && robot_y==5) || (old_state==3 && robot_x==5) || (old_state==4 && robot_y==0))

		return;

	else

		{

		scan_angle=660;

		//reset scanner

		servo_scan(scan_angle);

		delayMs(40);

		servo_scan(scan_angle);
```

```
delayMs(40);
servo_scan(scan_angle);
delayMs(40);

object_found=0;

//do a full scan until something is sensed in the way
while(scan_angle < 1000 && object_found==0)//was 942
        {
if (a2dConvert8bit(1) > scan_one_cell)//object found: add it to the map
                {
        //account for angle of the robot wrt the map, and stay in boundaries
                if (old_state==1)// && robot_x>0)
                        map[robot_x-1][robot_y]=wall;
                else if (old_state==2)// && robot_y<5)
                        map[robot_x][robot_y+1]=wall;
                else if (old_state==3)// && robot_x<5)
                        map[robot_x+1][robot_y]=wall;
                else if (old_state==4)// && robot_y>0)
                        map[robot_x][robot_y-1]=wall;
                object_found=1;
                }
```

```
            //servo scan code
            //-90 degrees at 300, lower goes CCW, 750 is centered, 1200 is far CW
            servo_scan(scan_angle);
            delayMs(12);
            scan_angle++;
            }
        if(object_found==0) //no object found, so clear it in the map
            {
            //account for angle of the robot wrt the map, and stay in boundaries
            if (old_state==1)// && robot_x>0)
                map[robot_x-1][robot_y]=nothing;
            else if (old_state==2)// && robot_y<5)
                map[robot_x][robot_y+1]=nothing;
            else if (old_state==3)// && robot_x<5)
                map[robot_x+1][robot_y]=nothing;
            else if (old_state==4)// && robot_y>0)
                map[robot_x][robot_y-1]=nothing;
            }
        }
//makes sure robot/goal location isnt replaced with a wall
//store robot location in map
map[robot_x][robot_y]=robot;
//store robot location in map
```

```
	map[goal_x][goal_y]=goal;
	}

//************DELAY FUNCTIONS************
//wait for X amount of cycles
void delay_cycles(unsigned long int cycles)
	{
	while(cycles > 0)
		cycles--;
	}
//***************************************

//-90 degrees at 300, lower goes CCW, 750 is centered, 1200 is far CW
void servo_scan(unsigned long int speed)
	{
	PORT_ON(PORTB, 1);
	delay_cycles(speed);
	PORT_OFF(PORTB, 1);//keep off
	//delayMs(8);
	}
```

WAVEFRONT ALGORITHM

```
//WAVEFRONT ALGORITHM

/*************************************************************************
*  Copyright (c) 2007 http://www.societyofrobots.com/programming_wavefront.shtml
*
*
*  This program is free software; you can redistribute it and/or modify
*  it under the terms of the GNU General Public License version 2 as
*  published by the Free Software Foundation.
*
*  Alternatively, this software may be distributed under the terms of BSD
*  license.
*
*         September 9th, 2007
*
*************************************************************************/
// Wave Front Variables
int nothing=0;
int wall=255;
int goal=1;
int robot=254;

//starting robot/goal locations
```

```
int robot_x=5;

int robot_y=2;

int goal_x=0;

int goal_y=4;

int cell_size=330;//the size of the robot

int scan_one_cell=39;//lower number makes it see further

//map locations

int x=0;

int y=0;

//temp variables

int counter=0;

//when searching for a node with a lower value

int minimum_node=250;

int min_node_location=0;

int reset_min=250;//anything above this number is a special item, ie a wall or robot

//X is vertical, Y is horizontal

int map[6][6]={{255,255,255,255,0,255},

                {255,0,0,0,0,255},

                {255,0,0,0,0,255},

                {255,0,0,0,0,255},

                {255,0,0,0,0,255},
```

```
{255,255,0,255,255,255}};

//declare functions here, first
int propagate_wavefront(int robot_x, int robot_y, int goal_x, int goal_y);
void unpropagate(int robot_x, int robot_y);

int min_surrounding_node_value(int x, int y);
void clear_map(void);

int propagate_wavefront(int robot_x, int robot_y, int goal_x, int goal_y)
        {
        //clear old wavefront
        unpropagate(robot_x, robot_y);


        counter=0;//reset the counter for each run!
  while(counter<50)//allows for recycling until robot is found
    {
    x=0;
    y=0;
        while(x<6 && y<6)//while the map hasnt been fully scanned
                {

                //if this location is a wall or the goal, just ignore it
                if (map[x][y] != wall && map[x][y] != goal)
                        {
```

```
                    //a full trail to the robot has been located, finished!
                    if (min_surrounding_node_value(x, y) < reset_min &&
map[x][y]==robot)
                            {
                            //finshed! tell robot to start moving down path
                            return min_node_location;
                            }
                    //record a value in to this node
                    else if (minimum_node!=reset_min)//if this isnt here, 'nothing' will go
in the location
                            map[x][y]=minimum_node+1;
                    }


            //go to next node and/or row
            y++;


            if (y==6 && x!=6)
                    {
                    x++;
                    y=0;
                    }
            }
            counter++;
    }
```

```
   return 0;
        }

void unpropagate(int robot_x, int robot_y)//clears old path to determine new path
        {
        //stay within boundary
        for(x=0; x<6; x++)
                for(y=0; y<6; y++)

                        if (map[x][y] != wall && map[x][y] != goal) //if this location is
something, just ignore it
                                map[x][y] = nothing;//clear that space

        //store robot location in map
        map[robot_x][robot_y]=robot;
        //store robot location in map
        map[goal_x][goal_y]=goal;
        }

//if no solution is found, delete all walls from map
void clear_map(void)
        {
        for(x=0;x<6;x++)
```

```
            for(y=0;y<6;y++)
                    if (map[x][y] != robot && map[x][y] != goal)
                            map[x][y]=nothing;
        }

//this function looks at a node and returns the lowest value around that node
//1 is up, 2 is right, 3 is down, and 4 is left (clockwise)
int min_surrounding_node_value(int x, int y)
        {
        minimum_node=reset_min;//reset minimum

        //down
        if(x < 5)//not out of boundary
                if  (map[x+1][y] < minimum_node && map[x+1][y] != nothing)//find the
lowest number node, and exclude empty nodes (0's)
                  {
                        minimum_node = map[x+1][y];
                        min_node_location=3;
       }

        //up
        if(x > 0)
                if  (map[x-1][y] < minimum_node && map[x-1][y] != nothing)
                  {
```

```
                minimum_node = map[x-1][y];
                min_node_location=1;
}

 //right
 if(y < 5)
        if  (map[x][y+1] < minimum_node && map[x][y+1] != nothing)
          {
                minimum_node = map[x][y+1];
                min_node_location=2;
}

 //left
 if(y > 0)
        if  (map[x][y-1] < minimum_node && map[x][y-1] != nothing)
          {
                minimum_node = map[x][y-1];
                min_node_location=4;
}

 return minimum_node;
 }
```

LANE KEEPER PROGRAM

```
/* Lane_Keeper.c
 * Designed to run on Create Command Module
 *
 * The basic architecture of this program can be re-used to easily
 * write a wide variety of Create control programs.  All sensor values
 * are polled in the background (using the serial rx interrupt) and
 * stored in the sensors array as long as the function
 * delayAndUpdateSensors() is called periodically.  Users can send commands
 * directly a byte at a time using byteTx() or they can use the
 * provided functions, such as baud() and drive().
 */

/*****************************************************************************
*       This program uses the robot's Front Left and Front Right Cliff Sensors to
*       detect a white line and follow it. Additionally, it uses the robot's IR
*       Receiver to detect signals coming from a pair of IR Emitters (or the
*       Standard Remote). When a signal is received, the robot changes its speed
*       accordingly (thus, reacting to an enforced "Speed Limit").
*
*       April 25th, 2007
*
*****************************************************************************/
```

```
// Includes
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/eeprom.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "oi.h"

// Constants
#define RESET_SONG 0
#define START_SONG 1
#define BUMP_SONG  2
#define END_SONG   3

// Global variables
volatile uint16_t timer_cnt = 0;
volatile uint8_t timer_on = 0;
volatile uint8_t sensors_flag = 0;
volatile uint8_t sensors_index = 0;
volatile uint8_t sensors_in[Sen6Size];
volatile uint8_t sensors[Sen6Size];
volatile int16_t distance = 0;
```

```
signed long int angle = 0;
volatile uint16_t update_delay = 16;//minimum of 15ms per sensor update allowed!!!
volatile uint8_t range = 0;

uint16_t        CliffSensorFL = 0;
uint16_t        CliffSensorFR = 0;
int j=0;

// Functions
void byteTx(uint8_t value);
void delayMs(uint16_t time_ms);
void delayAndUpdateSensors(unsigned int time_ms);
void initialize(void);
void powerOnRobot(void);
void baud(uint8_t baud_code);
void drive(int16_t velocity, int16_t radius);
void drive_distance(int16_t velocity, int16_t radius, int16_t distance_togo);
void drive_angle(int16_t velocity, int16_t radius, int16_t angle_togo);
void defineSongs(void);

//My Functions
void rotate_CCW(signed long int angle_tmp, signed long int velocity);
void rotate_CW(signed long int angle_tmp, signed long int velocity);
```

```
void straight(signed long int distance_tmp, signed long int velocity);
void stop(void);
void CheckIRRemote(void);

int main (void)
{
uint8_t leds_cnt = 99;
 uint8_t leds_on = 1;

// Set up Create and module
 initialize();
 LEDBothOff;
 powerOnRobot();
 byteTx(CmdStart);
 baud(Baud28800);
 defineSongs();
 byteTx(CmdControl);
 byteTx(CmdFull);

 // Stop just as a precaution
 drive(0, RadStraight);

 // Play the reset song and wait while it plays
 byteTx(CmdPlay);
```

```
 byteTx(RESET_SONG);
 delayAndUpdateSensors(750);

 while(timer_on)
 {
 CheckIRRemote();
 }

for(;;)
 {
        delayAndUpdateSensors(50);

  if(++leds_cnt >= 100)
  {
   leds_cnt = 0;
   leds_on = !leds_on;

   if(leds_on)
   {
    byteTx(CmdLeds);
    byteTx(LEDsBoth);
    byteTx(128);
    byteTx(255);
```

```
      LEDBothOff;
    }
    else
    {
      byteTx(CmdLeds);
      byteTx(0x00);
      byteTx(0);
      byteTx(0);
      LEDBothOn;
    }
  }

  delayAndUpdateSensors(10);

if(UserButtonPressed)
  {
    // Play start song and wait
    byteTx(CmdPlay);
    byteTx(START_SONG);
    delayAndUpdateSensors(2500);

          //reset encoder function
    angle = 0;
          distance = 0;
```

```
    while(!UserButtonPressed)

{
     if (j>0) //if interrupt occurs
            {
                    //Check for white line
                     if (CliffSensorFL>=600) //check on left
                    {
                    drive(100, 100); //drive and turn slightly to left
                    }
    else if (CliffSensorFR>=600) //check on right
                    {
                    drive(100, -100);//drive and turn slightly to right
                    }
    else
            {
            // Drive straight
            drive(100, RadStraight);
            }
            }
```

```
            delayAndUpdateSensors(50);
            drive(200,RadStraight);

        // Check for white line

         if (CliffSensorFL>=600) //check on left
                        {
                        drive(200, 100);
                        }
        else if (CliffSensorFR>=600) //check on right
                        {
                        drive(200, -100);
                        }
        else
                {
                // Drive straight
                drive(200, RadStraight);
                }

   }

        }
}
```

```
}

void CheckIRRemote(void) //interfaces with “speed limit” IR signal

{

        uint8_t IRchar;


        if(sensors[SenIRChar]==0xFF)

                return;


        IRchar = sensors[SenIRChar];


        switch (IRchar)

        {


        case IRRight:

                        j++;

                        break;


                case IRPause: //stops robot in its tracks

                  drive(0, RadStraight);

                        break;
```

```
                default:

                        break;

        }
        return;
}

// Serial receive interrupt to store sensor values
SIGNAL(SIG_USART_RECV)
{
 uint8_t temp;
temp = UDR0;
 if(sensors_flag)
 {
  sensors_in[sensors_index++] = temp;
  if(sensors_index >= Sen6Size)
   sensors_flag = 0;
 }
}

// Timer 1 interrupt to time delays in ms
SIGNAL(SIG_OUTPUT_COMPARE1A)
{
```

```
  if(timer_cnt)
    timer_cnt--;
  else
    timer_on = 0;
}

// Transmit a byte over the serial port
void byteTx(uint8_t value)
{
  while(!(UCSR0A & _BV(UDRE0))) ;
  UDR0 = value;
}

// Delay for the specified time in ms without updating sensor values
void delayMs(uint16_t time_ms)
{
  timer_on = 1;
  timer_cnt = time_ms;
  while(timer_on);
  }

// Delay for the specified time in ms and update sensor values
//Create updates its sensors and replies to sensor requests at a rate of
```

```
//67 Hz, or every 15 milliseconds. Do not send sensor requests faster than this.
void delayAndUpdateSensors(uint16_t time_ms)
{
 uint8_t temp;

 timer_on = 1;
 timer_cnt = time_ms;
 while(timer_on)
 {
        CheckIRRemote();
  if(!sensors_flag)
  {
   for(temp = 0; temp < Sen6Size; temp++)
    sensors[temp] = sensors_in[temp];

   // Update running totals of distance and angle
   distance += (int)((sensors[SenDist1] << 8) | sensors[SenDist0]);
   angle += (int)((sensors[SenAng1] << 8) | sensors[SenAng0]);
        byteTx(CmdSensors);
   byteTx(6);
   sensors_index = 0;
   sensors_flag = 1;
```

```
                    // distance from right wall
                CliffSensorFL=(sensors[SenCliffFLSig1]<<8)|sensors[SenCliffFLSig0];
                CliffSensorFR=(sensors[SenCliffFRSig1]<<8)|sensors[SenCliffFRSig0];
   }

 }
 return;
}

// Initialize the Command Module's ATmega168 microcontroller
void initialize(void)
{
 cli();

 // Set I/O pins
 DDRB = 0x12;          //12, 10   //0b00010010 (4 must be output, 5 must be input)// use a
1 for servos
 PORTB = 0xCF; //CD, CF   //0b11001111
 DDRC = 0x00;  //configure all C ports for input
 PORTC = 0x00; //make sure pull-up resistors are turned off
 DDRD = 0xE6;  //0b11100110 (1's for input)
 PORTD = 0x7D; //0b01111101 (1's for output)

 // Set up timer 1 to generate an interrupt every 1 ms
```

```
  TCCR1A = 0x00;
  TCCR1B = (_BV(WGM12) | _BV(CS12));
  OCR1A = 71;
  TIMSK1 = _BV(OCIE1A);

  // Set up the serial port with rx interrupt
  UBRR0 = 19;
  UCSR0B = (_BV(RXCIE0) | _BV(TXEN0) | _BV(RXEN0));
  UCSR0C = (_BV(UCSZ00) | _BV(UCSZ01));

  /* initialize I2C (TWI) clock: 100 kHz clock, TWPS = 0 => prescaler = 1 */
  TWSR = 0;
  TWBR = (F_CPU / 100000UL - 16) / 2;

// Turn on interrupts
  sei();
}

void powerOnRobot(void)
{
  // If Create's power is off, turn it on
  if(!RobotIsOn)
  {
```

```
    while(!RobotIsOn)
    {
       RobotPwrToggleLow;
       delayMs(500);  // Delay in this state
       RobotPwrToggleHigh;  // Low to high transition to toggle power
       delayMs(100);  // Delay in this state
       RobotPwrToggleLow;
    }
    delayMs(3000);  // Delay for startup
 }
}

// Switch the baud rate on both Create and module
void baud(uint8_t baud_code)
{
 if(baud_code <= 11)
 {
   byteTx(CmdBaud);
   UCSR0A |= _BV(TXC0);
   byteTx(baud_code);
   // Wait until transmit is complete
   while(!(UCSR0A & _BV(TXC0))) ;
```

```
cli();

// Switch the baud rate register
if(baud_code == Baud115200)
  UBRR0 = Ubrr115200;
else if(baud_code == Baud57600)
  UBRR0 = Ubrr57600;
else if(baud_code == Baud38400)
  UBRR0 = Ubrr38400;
else if(baud_code == Baud28800)
  UBRR0 = Ubrr28800;
else if(baud_code == Baud19200)
  UBRR0 = Ubrr19200;
else if(baud_code == Baud14400)
  UBRR0 = Ubrr14400;
else if(baud_code == Baud9600)
  UBRR0 = Ubrr9600;
else if(baud_code == Baud4800)
  UBRR0 = Ubrr4800;
else if(baud_code == Baud2400)
  UBRR0 = Ubrr2400;
else if(baud_code == Baud1200)
  UBRR0 = Ubrr1200;
```

```
    else if(baud_code == Baud600)
      UBRR0 = Ubrr600;
    else if(baud_code == Baud300)
      UBRR0 = Ubrr300;

    sei();

    delayMs(100);
  }
}
// Send Create drive commands in terms of velocity and radius
void drive(int16_t velocity, int16_t radius)
{
  byteTx(CmdDrive);
  byteTx((uint8_t)((velocity >> 8) & 0x00FF));
  byteTx((uint8_t)(velocity & 0x00FF));
  byteTx((uint8_t)((radius >> 8) & 0x00FF));
  byteTx((uint8_t)(radius & 0x00FF));
}

// Define songs to be played later
void defineSongs(void)
{
  // Reset song
```

```
byteTx(CmdSong);
byteTx(RESET_SONG);
byteTx(4);
byteTx(60);
byteTx(6);
byteTx(72);
byteTx(6);
byteTx(84);
byteTx(6);
byteTx(96);
byteTx(6);

// Start song
byteTx(CmdSong);
byteTx(START_SONG);
byteTx(6);
byteTx(69);
byteTx(18);
byteTx(72);
byteTx(12);
byteTx(74);
byteTx(12);
byteTx(72);
```

```
byteTx(12);
byteTx(69);
byteTx(12);
byteTx(77);
byteTx(24);

// Bump song
byteTx(CmdSong);
byteTx(BUMP_SONG);
byteTx(2);
byteTx(74);
byteTx(12);
byteTx(59);
byteTx(24);

// End song
byteTx(CmdSong);
byteTx(END_SONG);
byteTx(5);
byteTx(77);
byteTx(18);
byteTx(74);
byteTx(12);
```

```
 byteTx(72);
 byteTx(12);
 byteTx(69);
 byteTx(12);
 byteTx(65);
 byteTx(24);
}

//rotate a certain angle - 100 = 90 degrees!
void rotate_CCW(signed long int angle_tmp, signed long int velocity)
        {
        //Counter-clockwise angles are positive and clockwise angles are negative.
        drive(velocity, RadCCW);

        while(angle<angle_tmp)
                delayAndUpdateSensors(update_delay);//flaw is that if it goes above in the
middle of function, it will drift

        stop();
        delayAndUpdateSensors(update_delay);

        angle=angle-angle_tmp;//resets angle to 0; accounts for an over/undershoot
        angle=0;
```

```
	}
//rotate a certain angle - 100 = 90 degrees!
void rotate_CW(signed long int angle_tmp, signed long int velocity)
	{
	//Counter-clockwise angles are positive and clockwise angles are negative.
	drive(velocity, RadCW);

	while((-angle)<angle_tmp)
		delayAndUpdateSensors(update_delay);//flaw is that if it goes above in the middle of function, it will drift

	stop();
	delayAndUpdateSensors(update_delay);

	angle=angle-angle_tmp;//resets angle to 0; accounts for an over/undershoot
	angle=0;
	}


//drive straight function (use negative velocity to go in reverse)
void straight(signed long int distance_tmp, signed long int velocity)
	{
	drive(velocity, RadStraight);
```

```
        while(distance<distance_tmp)//flaw is that if it goes above in the middle of
function, it will drift

                delayAndUpdateSensors(update_delay);



        stop();

        delayAndUpdateSensors(update_delay);



        distance=distance-distance_tmp;//resets angle to 0; accounts for an
over/undershoot

        }


//stop motors

void stop(void)

        {

        drive(0, RadStraight);

        delayAndUpdateSensors(update_delay);

        }
```

ARDUINO: TRAFFIC LIGHT

```
int IR_PIN = 13;
int led_pin = 13;
int red = 11;
int yellow = 10;
int green = 9;
int bin_1 = 3000;       //Binary 1 threshold (Microseconds)
int bin_0 = 1000;       //Binary 0 threshold (Microseconds)

#define BIT_IS_SET(i, bits)  (1 << i & bits)

const int period = 26;       // this is the period of 38kHz, in microseconds
const int PULSE_WIDTH = 10;   // time between pulses, in microseconds

// some basic codes taken out of the manual:
const int FORWARD = 130;
const int PAUSE = 137;

void setup() {
  pinMode(led_pin, OUTPUT);
  pinMode(IR_PIN, OUTPUT);
  pinMode(red, OUTPUT);
    pinMode(yellow, OUTPUT);
      pinMode(green, OUTPUT);
  digitalWrite(led_pin, LOW);
  digitalWrite(IR_PIN, LOW);
  digitalWrite(red, LOW);
```

```
  digitalWrite(yellow, LOW);
   digitalWrite(green, LOW);
 Serial.begin(9600);
}

void loop() {

 // lights LEDs and sends stop or go command, respectively
 digitalWrite(yellow, HIGH);
 delay(3000);
   digitalWrite(yellow, LOW);
  digitalWrite(red, HIGH);
 send_byte(PAUSE);
 delay(5000);
 digitalWrite(red, LOW);
   delay(5000);
  digitalWrite(green, HIGH);
 send_byte(FORWARD);
 delay(5000);
   digitalWrite(green, LOW);



}

// oscillates at 38kHz to write "ON"
void on(int time) {
```

```
  for(time = time/period; time > 0; --time) {
    digitalWrite(IR_PIN, HIGH);
    delayMicroseconds(PULSE_WIDTH);

    digitalWrite(IR_PIN, LOW);
    delayMicroseconds(PULSE_WIDTH);
  }
}

// just write pin low to write "OFF"
void off(int time) {
    digitalWrite(IR_PIN, LOW);
    delayMicroseconds(time);
}

// write a binary 1
void WriteOne() {
  on(bin_1);    // on for 3ms
  off(bin_0);   // off for 1ms
}

// write a binary 0
void WriteZero() {
  on(bin_0);    // on for 1ms
  off(bin_1);   // off for 3ms
}
```

```
// Send a byte over the IR LED
void send_byte(int bits) {
  for (int i = 7; i >= 0; i--)
  {
    if (BIT_IS_SET(i, bits)) {
      WriteOne();
    } else {
      WriteZero();
    }
  }
  delay(4);      // leave 4ms at the end of the instruction
}
```

ARDUINO: SPEED LIMIT

```
int IR_PIN = 13;
int led_pin = 13;
int bin_1 = 3000;       //Binary 1 threshold (Microseconds)
int bin_0 = 1000;       //Binary 0 threshold (Microseconds)

#define BIT_IS_SET(i, bits)  (1 << i & bits)

const int period = 26;       // this is the period of 38kHz, in microseconds
const int PULSE_WIDTH = 10;   // time between pulses, in microseconds

// some basic codes taken out of the manual:
const int RIGHT = 131;

void setup() {
  pinMode(led_pin, OUTPUT);
  pinMode(IR_PIN, OUTPUT);
  digitalWrite(led_pin, LOW);
  digitalWrite(IR_PIN, LOW);
  Serial.begin(9600);
}

void loop() {

  // Sends IR command to slow robot's speed
  delay(3000);
  send_byte(RIGHT);
```

```
}

// oscillates at 38kHz to write "ON"
void on(int time) {

  for(time = time/period; time > 0; --time) {
    digitalWrite(IR_PIN, HIGH);
    delayMicroseconds(PULSE_WIDTH);

    digitalWrite(IR_PIN, LOW);
    delayMicroseconds(PULSE_WIDTH);
  }
}

// just write pin low to write "OFF"
void off(int time) {
    digitalWrite(IR_PIN, LOW);
    delayMicroseconds(time);
}

// write a binary 1
void WriteOne() {
  on(bin_1);    // on for 3ms
  off(bin_0);   // off for 1ms
}
```

```
// write a binary 0
void WriteZero() {
  on(bin_0);    // on for 1ms
  off(bin_1);   // off for 3ms
}

// Send a byte over the IR LED
void send_byte(int bits) {
  for (int i = 7; i >= 0; i--)
  {
    if (BIT_IS_SET(i, bits)) {
      WriteOne();
    } else {
      WriteZero();
    }
  }
  delay(4);      // leave 4ms at the end of the instruction
}
```

APPENDIX D: PARTS AND COSTS

Table II. Costs Sheet

Item	Cost	Source	Purpose
iRobot Create® Premium Development Package	$299.99	iRobot Store	Automated Vehicle
Sharp IR Rangefinder	$13.99	RobotShop	Obstacle Detection/Avoidance
Rangefinder Cable	$1.95	RobotShop	For Rangefinder
HS-311 Servo Motor	$8.99	RobotShop	Rotates Rangefinder
Arduino Uno Boards	$57.90	RobotShop	Traffic Signal Transmitters
IR Emitters	$2.92	Newark	For Arduino Boards
Foam Mat	$59.99	Overstock	Road Surface
Fabric Boxes	$132.81	Staples	Obstacles
PC Board	$1.99	RadioShack	For Soldering Electronics
Jumpers & Headers	$7.50	RobotShop	For Connecting Electronics
9V Power Adapters	$37.38	RadioShack	For Arduino Boards
Colored LEDs	$4.47	RadioShack	Traffic Light
Resistors	$0.99	RadioShack	For LEDs
		TOTAL:	**$630.87**

APPENDIX E: TIMELINE

Table III. Senior Design Phase I Timeline

AUG	SEPT	OCT	NOV	DEC
Formed Team	Chose Project Topic	Created Project Outline	Project Proposal	Project Presentation
	Chose Advisor	Purchased iRobot Create®	Practiced with Hardware & Software	Research
	Research	Studied Robot Manuals	Final Report Submitted	First Parts Order
	Chose Robot Platform			

Table IV. Senior Design Phase II Timeline

JAN	FEB	MAR	APR	MAY
Attached Rangefinder & Tested	Tested Arduino	Robot Receives IR Signals	Assembled Physical Map	Project Presentation
Research	Uploaded Wavefront	Received Approval for Grants	Success with "Traffic Light"	
Applied for Grants		Research	Trial Runs with Main Program	
Second Parts Order		Tested Cliff Sensors	Successful Trial Run	
		Third Parts Order	Final Report	