Joshua Burkhart

12/12/2010

Dr. Robert Adams

CS467

Autonomous Vehicle: Final Report

The goal of the Autonomous Vehicle project was to develop a rover that could navigate to predetermined GPS coordinates and avoid obstacles without human interaction. The Autonomous Vehicle project is the result of integrating several major components. The arduino micro controller development device was essential to our project. It gave us the ability to control other devices with uploaded programs. The motor shield allowed us an interface between the DC motors used for transportation and the programs loaded on the arduino. It also gave us the ability to control voltage more precisely through the use of existing code libraries. The electronic compass gave us the ability to reliably and accurately calculate our bearing agnostic of GPS signal quality. The optoelectronic device gave us the ability to detect distance with the use of an IR emitter and receiver. This was used for collision avoidance. One part of the project I would have liked to have changed was the chassis. Though not very energy-hungry, the small size of our chassis prevented us from outdoor testing. The final major part of our project was the software. Due to the nature of the arduino architecture, the size of the code and memory footprint had to be constantly reduced (he flash memory available was limited to 14KB and the ram was limited to 1K). This brought into practice the use of many global variables into an otherwise modular program design. On another note, the ability to reference existing libraries allowed for fairly easy device manipulation. Finally, the GPS module that allowed for reading of the rover's current coordinate. Though this feature was not fully integrated with the vehicle, it was tested over a serial connection and confirmed to be working. It is estimated that with a small amount of effort such a device could complete the Autonomous Vehicle project.

The arduino is a micro controller development environment used to integrate various pieces of hardware and manipulate them through the program loaded onto it.

The program is written in what is called the 'processing' language and resembles C. Many libraries are freely available for the arduino and compatible hardware components making everything from motor control to serial communication more intuitive. The code is organized into two rudimentary parts. First is what is called the 'Setup()' function. This will run once when the arduino is powered on and is typically used to set the values of global variables. The second is called the 'Loop()' function. After the 'Setup()' function is called, t will be called continuously so long as the arduino has power. It is here that the control logic of the device must be.

The arduino hardware is most easily accessed through the use of its pins. Pins allow for compatible devices to access things like 3v VDD (power), 5v VDD (power), and GND (ground), as well as both digital and analog pins for interacting with the arduino itself. Digital pins made serial communication possible while analog pins were useful for reading voltages. Both types of pins were essential to our project. We used digital pins when interfacing with the motor shield and analog pins when interfacing with both the digital compass and the optoelectronic device.

The arduino IDE is an additional feature of the arduino development environment. It lacks in comfy features like auto-completion and repository integration but simplifies the process of compiling code, uploading code through the arduino serial connection, and serial monitoring.

The motor controller we used for the project was provided by adafruit industries. It allows for precise control of motor position, velocity, and acceleration of one or more electric motors. In the case of this project we used small DC motors, though the device is capable of operating larger stepper motors. One largely beneficial feature of the device was its preexisting code library compatible with arduino software. Using this library to generate our own wrapper functions, it became trivial to move the device forward or backward or to turn it left or right.

Though we purchased the Hitachi HM55B Dual-Axis Compass Module, we received a

Honeywell model with identical specifications and schematic (this did not have any noticeable effect on the project but is interesting to note). Unlike the motor shield mentioned above, this compass is not particularly made for the arduino and therefore did not come with an easy-to-use library. Because of this, functions parsing the compass's output were required to calculate bearing. After specifications were carefully studied and many forums consulted on integrating this device with the arduino, appropriate functions were written and tested to be working. The compass could give direction as a number from 0-360 (indicating degree). It would have been ideal to map this directly to a unit circle with each increment of 90 being equivalent to one of either North, South, East, or West. This was not possible for two main reasons. First, the compass was susceptible to interference by electronic forces (electric motors). This was effect was reduced by raising the compass up from the motors several inches. Second, the compass required it was parallel with the surface it was on. This was more difficult to achieve than would be expected as the vehicle tended to 'sit back' in its tracks while moving forward, thus slightly tilting the compass. Correcting for this would have of course had adverse effects on the compass reading while the vehicle was standing still or turning. The only feasible way to reduce this effect would have been to use a gimbal, a device that was not in the project's budget. Because of these two effects, experimental values were used for North, South, East, and West (usually deviating from ideal values by a matter of a few percent). Communication with the compass was achieved by receiving and manipulating bytes sent to the arduino's analog pins. The code for this operation is viewable in our source (https://github.com/joshuagburkhart/Autonomous-Vehicle-GVSU).

The optoelectronic device used in the project was the Sharp GP2Y0A21YK. Specifications state the range as 10cm to 80cm with the threshold for our project set at an experimental value of approximately 18in or 45cm. The device uses an emitter and a receiver of infrared radiation to detect distance and outputs a voltage indirectly proportional to that distance (high voltage = small distance). Reading the output of the device was made simple by using the analog pins of the arduino. Though this device was not specifically made for the arduino and has no corresponding libraries (like the compass),

the usage and integration of the device was simple (unlike the compass). The device was mounted on one side of the vehicle chassis and does a good job of detecting objects directly in front of it. Because it is mounted toward one side, the track on the far side can still run into objects. This could be avoided by mounting a second optoelectronic device on that side. The software uses the device by checking the voltage reading often (about every 50ms), to assure it is below the experimental threshold. The 50ms delay is appropriate as the device specification states 32ms as its response time. On the event that the voltage reading comes back above the experimental threshold, the program reverses both motors and begins to pivot the vehicle, scanning for a low voltage reading (indicating a clear path).

The chassis of the vehicle is one component that did not work out as well as planned. Most probably, a more functional and less expensive alternative would have been to use the chassis of a radio controlled car with larger motors, a larger power supply, and higher ground clearance. The chassis we used provided us with bidirectional DC motors. The motors were mounted on the bottom of the chassis with no skid plate preventing moisture or grime from collecting on them during use. The connections for these motors were uncovered and could have easily been compromised during outdoor use. The frame of the vehicle was small and did not provide enough room for the 12v battery that was desired for the project. Instead an array of parallel 9v batteries was constructed to satisfy energy needs. The chassis did come with an almost useless mount for four AA batteries in series. Useless because four AA batteries produce 6v when wired in series. The recommended input voltage of an arduino is 7-12v. The chassis did not provide many mounting options for the arduino or other hardware used for the project. The vehicle tracks were small and provided little ground clearance. They were constructed of hard plastic and made obstacles out of things as small as the edge of a carpet.

The production software integrates the arduino with the above devices. Its general structure consists of several loops causing certain conditions to be constantly checked. One example is voltage readings outputted by the optoelectronic device. Another assures the compass's heading is within a certain tolerance. A more complicated issue solved by the software is the matter of calculating a

bearing from a given a current heading, a current latitude / longitude, and a desired latitude / longitude. This is solved with the use of a '%' wrapper function and the 'atan2()' function. The wrapper function was required because the '%' function naturally will return negative values when passed negative values. This does not map well onto a unit circle and is not what a modulus operation should return. The wrapper function (called 'mod()') assures that only positive numbers are returned. The 'atan2()' function gives the degree in radians of a coordinate from the x-plane. This value is converted to degrees and mapped to the compass orientation. All of the software for this project is well documented and freely browseable in our repository online

The GPS unit, though not fully integrated into the project was an impressive piece of hardware. It could receive and output its coordinates in NMEA format over a serial connection. It is believed that without much effort, the NMEA format could be converted into a latitude and longitude. The device would then need to be connected to the rest of the functioning hardware and continue to provide the vehicle with current GPS coordinates.

3.02. Ensure proper and achievable goals and objectives for any project on which they work or propose.

The goals of this project were clearly stated at the beginning of the project and were achieved in a very similar order as predicted. Though not all of the desired goals were eventually realized, I believe they were all realistically attainable.

3.07. Strive to fully understand the specifications for software on which they work.

Much of the work devoted to this project was in the form of research. Due to the team members' unfamiliarity of the domain, this was appropriate. Research of both hardware components and their libraries were the most common research topics. I feel that at the conclusion of the project, at least 50% of the team members have gained a substantial amount of subject knowledge.

3.11. Ensure adequate documentation, including significant problems discovered and solutions adopted, for any project on which they work.

During the course of the project, work was well documented. This can be seen in many ways. Code modifications were tracked using a resource called Github.com, team member hours were tracked using a resource called toggl.com, and general goals / difficulties were discussed in several milestone reports.

3.13. Be careful to use only accurate data derived by ethical and lawful means, and use it only in ways properly authorized.

This project used free software resources exclusively. Though hardware became a relatively large expense, resources such as the compiler, the IDE, the code repository, and the time tracking tool were without cost. Using free resources this project encourages technological growth by promoting and displaying the abilities of accessible tools.

In conclusion, this project gave the opportunity to learn several new skills. Learning to interface with hardware was very rewarding. Not only was it completely new and unfamiliar, but something that students of the GVSU computer science program seldom have the opportunity to do.

Difficulties with our system stemmed largely from inexperience with the hardware used in the project. Some programming problems were encountered and addressed while working on the navigation logic, but were quickly resolved.

There were many things about the project that could have worked out better than they did. We could have had a larger budget. This would have allowed us to purchase better and more precise hardware components. Components desired include an ultrasonic sensor, a larger chassis, and a gimbal for the electronic compass. We could have had more members on the team. This would have allowed the work to be more evenly distributed among team members and prevent project failure in the chance

that one of the team members fell behind in his/her responsibilities. We could have had a poster at the GVSU project day. This would have given us the opportunity to better present our project to the public.

Possible extensions to the project are vast. Obvious extensions could include additional range-finding hardware to improve collision avoidance, a covered chassis with higher ground clearance to allow for outdoor testing, and a more accurate GPS unit.