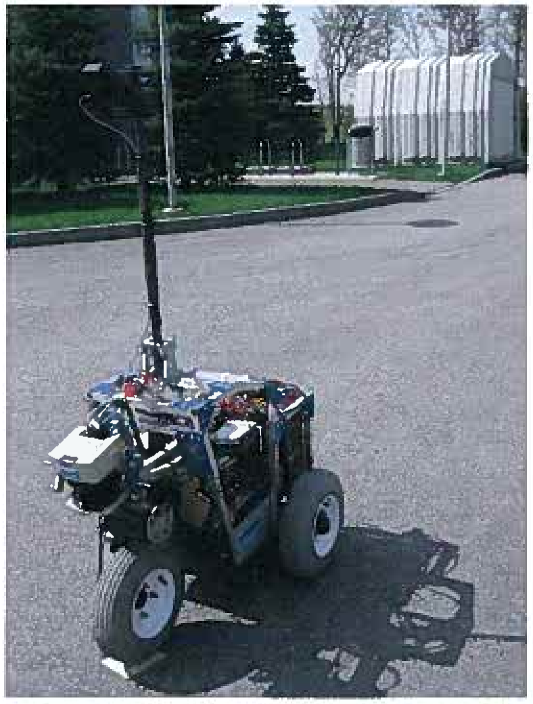

Brian

Franklin W. Olin College of Engineering
2009 Intelligent Ground Vehicles Competition Entry

Nicholas Hobbs, Daniel Grieneisen, Jacob Izraelevitz,
Arash Ushani, & Raphael Cherney

Faculty Advisor: Professor David Barrett

Required Faculty Statement: I certify that the engineering design of the new vehicle, Brian, described in this report, has been significant and equivalent to what might be awarded credit in a senior design course.

David Barrett

Professor David Barrett

Contents

1 Introduction

This paper describes the 2009 Franklin W. Olin College of Engineering Intelligent Ground Vehicles Competition entry: *Brian*. *Brian* is an intelligent ground vehicle specializing in road following, GPS navigation, and obstacle avoidance. The team at Olin designed, constructed, and tested *Brian* starting in February 2009 and ending May 16th, 2009. This paper describes the vehicle's design as well as the methodologies used to develop it.

2 Team Organization

The Olin College IGVC team has five members, all of whom are sophomores. There are only two formalized positions: team captain and financer. The team captain manages team progress while the financer tracks the team's expenditures and completes necessary paper work. Given the short time frame in which the team had to complete the design and implementation of the robot, we could not make a full recruiting effort. However, we are able to leverage the talents of our team and our flat organizational structure to quickly fulfill objectives. The team is composed of two electrical engineers, one mechanical engineer, one systems engineer, and one software engineer. Using these natural talents, we are able to distribute tasks in order to meet major milestones. We use a work in progress – or WIP – method to distribute labor. On a two week basis we define a list of sub-tasks whose completion fulfills those weeks overall objectives. We then assign each task to a specific team member, who places his or her tasks on our lab's WIP board. Each person has three task categories on the board: to do, in progress, and complete. This aid allows the team leader and members to monitor progress and ensure that work is distributed evenly throughout the semester. It also clearly defines expectations and responsibilities for each team member.

3 Design Process

In order to make design decisions, our team follows an iterative process. Because of our relative inexperience in building intelligent vehicles, we begin each major decision with a concerted research effort. Each team member reads papers from previously successful IGVC teams and other robotics research universities. We then compare possible solutions in a decision matrix that compare each idea's merits on their effectiveness at achieving their goal and their ease of implementation. Based upon this comparison, a method is adopted and a unique metric for testing that solution's effectiveness is developed. In order to

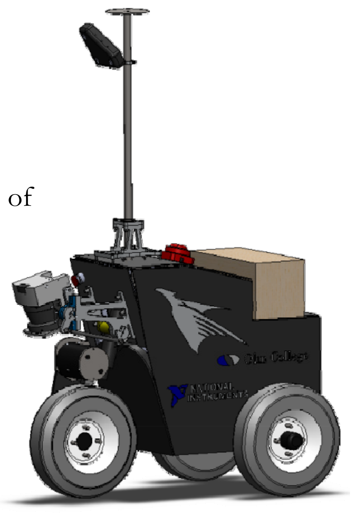
Figure 1: The final SolidWorks model of *Brian*.

implement the proposed design, we begin with the creation of a computer simulation and then make physical prototypes. Simulation is done using SolidWorks and Comsol for mechanical design and LabVIEW for software development. After implementation, if our approach passes required benchmark testing, we move on. If not, a new solution was implemented and tested. To augment this iterative process, our team also holds community-wide design reviews each month to seek feedback from professors and students. This invaluable, outside feedback leads to major design revisions that improve robot function.

In this iterative process we make decisions based on the following set of values, defined at the beginning of the season.

- *Brian* will serve as research platform for other students in Olin Intelligent Vehicles Lab as well as our future IGVC mechanical platform. As a result, our highest priority was **creating a simple, robust robot that would allow for extensive future software development**.
- Second, we hoped to **minimize the cost of robot production.** To do this we mostly used components our college already owned or parts donated by sponsors.
- Finally, we wanted to **create a design that could be implemented in four months**, allowing us to qualify for the 2009 IGVC. As a result, we favored designs that we could implement quickly, allowing ample time for systems integration, debugging, and testing.

These guiding values and our design processes led to the hardware and software decisions that are outlined in sections 4 and 5 of this document.

4 Hardware

The 2009 Olin College IGVC entry, *Brian*, incorporates several innovations in both selection and implementation of hardware. The hardware design adopted by the Olin IGVC Team is outlined in this section. First, we introduce a basic description of *Brian's* physical systems. Brian is a retrofitted *Chimp* Human Transport Vehicle from Doran Electric Vehicles. The only remaining systems from the original vehicle are the undercarriage frame, the drive wheels, the axles, and the front drive motor. All other fabrication and design were done in house at Olin College. The chassis is made of 6020 aluminum square tubing and plate and connected using galvanized steel mounting brackets. The entire system weighs three-hundred-sixty pounds. All of the vehicle's major components are listed in the following bill of expenditures.

Part	Cost to Team	Estimated Cost
Point Grey Digiclops Stereovision System	$0	$9000
SICK LMS291-S05 LIDAR	$0	$6000

Doran Electric Vehicles Chimp Human Transport[1]	$0	$3195
Dell Latitude D620	$0	$2000
SmallPC SC525-P24	$0	$1300
Sensoray 626 Analog & Digital I/O	$0	$660
Ublox Odometry Enabled GPS	$0	$600
Galil MSA-12-80 Motor Controller (quantity 2)	$0	$360
Pittman GM14904S016-R1Motor with Encoder	$350	$350
Autec Wireless Emergency Stop	$0	$340
Fastcom 422/2-ISA	$0	$300
Pitman GM9236S027-R1	$260	$260
Mechanical Emergency Stop Button (quantity 2)	$240	$240
Weatherproof Encoder	$0	$225
Aluminum Stock	$180	$180
Rosewill RC-602 Firewire Card	$0	$15
Total	**$1,030**	**$25,025**

4.4 Hardware Innovations

Hardware innovations on *Brian* come from both the selection of components and their utilization. When selecting sensors for *Brian* the Olin IGVC Team created redundant systems that more robustly avoid a wider range of obstacles. By using both the Point Grey System and SICK LIDAR to detect obstacles, *Brian* compensates for both sensors' points of failure.

The other major hardware innovation was how these sensors are mounted. To begin, we examine the mounting, and resulting actuation, of the Point Grey System. The camera mounts to a seventy-inch sensor mast. However, instead of statically fusing the mast to the chassis, it is tied to the steering column with a universal joint, as shown in Figure 2. The universal joint allows two key advantages. First, it points the camera in the same direction that the front wheel points. The camera has a narrow ninety degree viewing angle, while the robots front wheel can move to a steering angle of forty five degrees. If the camera were statically mounted, the robot would only see obstacles and lines to the inside of the wheel when taking steep, forty-five degree turns. Thus, by rotating the camera, *Brian* always sees the maximum viewing area of the ground he is about

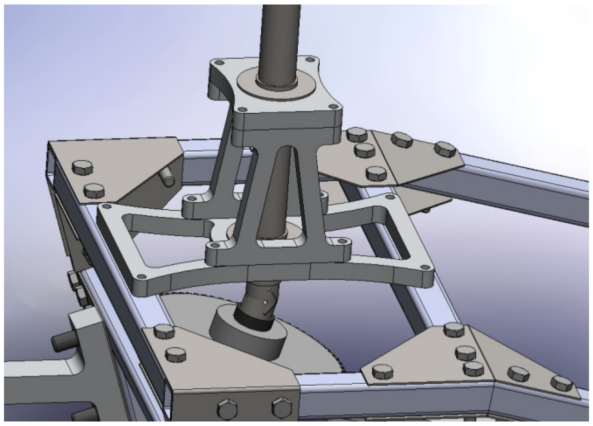
Figure 2: The universal joint used to turn our camera to face towards the front wheel's direction of travel.

[1]Includes two Chairman AGM-12100T Deep Cycle Lead Acid Batteries and one Curtis 1228 controller

to cross over. We also use the universal joint to gain another key advantage. This joint transforms the camera's axis of rotation to be perpendicular to the ground plane. This both makes our transforms easier to understand, and means we have more consistent camera data. Thus, by tying the camera to the front wheel's actuation via the universal joint, we are able to better detect lines and obstacles during vehicle turns.

The other primary outward facing sensor, the LIDAR, is also actuated. The LIDAR's placement provides an ability key to this year's competition: pothole detection. By mounting the sensor facing down toward the ground plane, we are able to detect the two inch potholes that must be avoided as per 2009 IGVC rules. Additionally, by actuating the LIDAR head, we give *Brian* a higher level of function. The actuated LIDAR can be used to adjust our scanning angle during a competition run. This allows *Brian* to increase or decrease his scanning range based on proximity to desired target. In the future, the addition of oscillation will make *Brian* a more flexible research platform. Actuating the head allows rapid testing of different head positions. This also allows for continuous oscillation. When combined with higher processing speeds, this motion can yield a three dimensional LIDAR image.

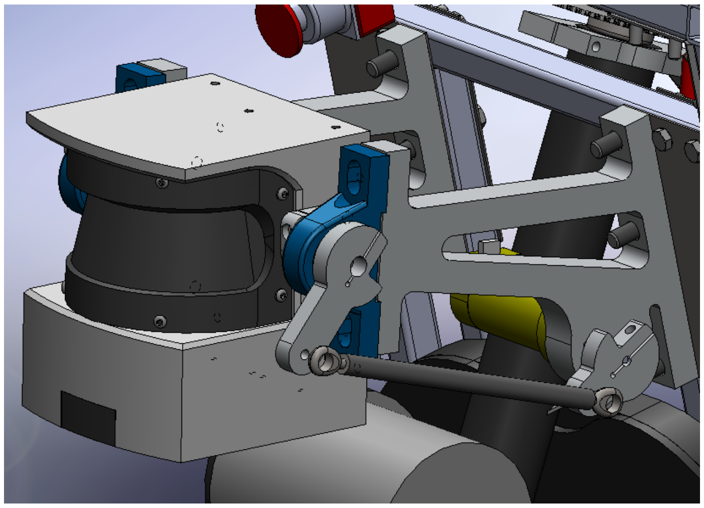
Figure 3: The four bar linkage used to actuate the LIDAR.

Additionally, the method of oscillating the LIDAR is advantageous. The system uses a four bar linkage. As a result, constant rotation of the attached motor in one direction simply oscillates the LIDAR head up and down through a range of adjustable values. This both makes constant oscillation easier, while also ensuring that software failure does not crash this expensive sensor into the chassis. By using a four bar linkage, the LIDAR has a fail safe mode and can oscillate without reversing motor direction.

4.2 Vehicle Dynamics & Encoder Localization

To understand *Brian's* motion, we use the model of vehicle dynamics outlined in this section. Brian is a human transport tricycle with a single velocity-controlled front drive wheel. The front drive wheel's steering angle actuation is position controlled using a Pittman motor. As a result, *Brian's* motion is nonholonomic. To describe this nonholonomic motion, we developed a mathematical model, whose constants are defined in Figure 4. The robot's change in global heading, Δφ, can be modeled by

$$\Delta\varphi = \frac{\Delta S \sin(\theta)}{L}$$

where ΔS is the change in distance traveled by the front wheel, θ is the steering angle of the vehicle, and L is the distance between the front and rear axle of the vehicle. Using this, we can determine the distance traveled over time

$$d = r\sqrt{2 - 2\cos(\Delta\varphi)}$$

where r is the turning radius of the vehicle.

Finally, from this we derive the change in global position as:

$$\Delta x = \cos\left(\varphi + \frac{\Delta\varphi}{2}\right)$$

and

$$\Delta y = d\sin\left(\varphi + \frac{\Delta\varphi}{2}\right)$$

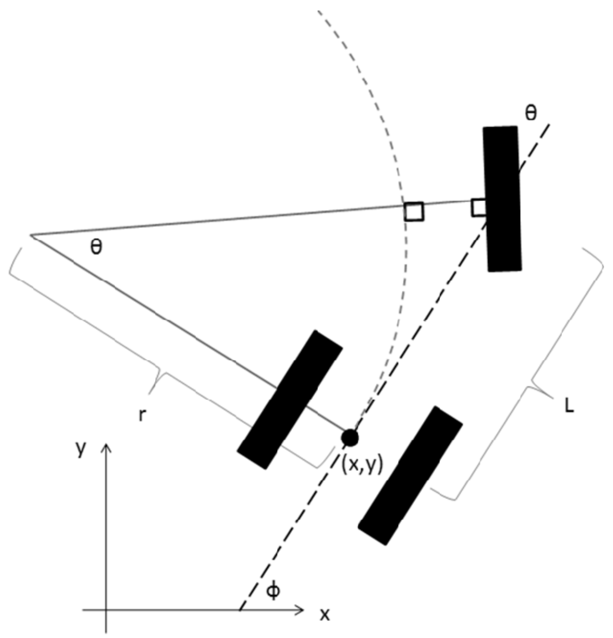

Figure 4: Measurements that are used to describe Brian's vehicle dynamics: the steering angle is θ, the turning radius is *r*, the distance between rear and front axle is L, the robot heading within the global reference frame is φ, and the global position (from the center of the rear axle) is (*x*,*y*).

where Δx and Δy are the change of the robots global position in the Cartesian coordinates (x,y). With this information, we have both a mathematical model that proactively controls robot motion and retroactively determines the robot position. This model converts wheel odometry to a precise robot localization over relatively small distances.

4.3 Sensors

The sensors onboard *Brian* are divided into three separate classes: localization, object detection, and line detection. *Brian* localizes using five sensors: three quadrature encoders, one Hall effect sensor, and one ublox GPS. The encoders measure rotational position of the motors powering the LIDAR tilt, drive wheel, and the steering angle. These values can be used to estimate vehicle position based on the mathematical models outlined in section 4.2. However, these sensors only measure change in position, not absolute position. As such, they must begin properly zeroed. The mechanical design of the LIDAR head allows for easy manual zeroing, as the four bar linkage gives the LIDAR an obvious minimum position. The drive motor does not require zeroing, as it is velocity controlled. The steering angle control motor requires both the most accurate zero and has the most difficult to find zero position. To mitigate error generated by inaccurate steering motor zeroing, we installed a Hall effect sensor. The Pittman motor used to position control *Brian*'s steering angle connects directly to a large sprocket that turns the steering column via a chain linkage. The Hall effect sensor is mounted directly below this sprocket. This, in concert with a sprocket-mounted magnet, allows *Brian* to accurately zero his front motor position which in turn creates more accurate odometry localization.

In addition to localizing using encoder informed wheel odometry, *Brian* uses a ublox Encoder Enabled GPS to localize globally. This unit has a relatively cheap GPS chip. However, it does have one key advantage. The system not only localizes using readings from GPS satellites, it also incorporates odometry from wheel encoders and an onboard accelerometer. These sensor inputs are synthesized in the ublox using a Kalman Filter to create a single localization. This both increases the accuracy of the unit and keeps the unit from losing localization during periods of GPS outage.

Brian detects objects using two outward facing sensors: the SICK LMS291-S05 LIDAR and the Point Grey Digiclops Stereovision System. The SICK uses the flight time of emitted 905nm ultraviolet light pulses to calculate the distance to nearby objects. It returns a distance to nearest obstacle at every one degree increment in the range of angles shown in Figure 5**.** The instrument is currently calibrated to measure obstacles at a range of up to thirty-two meters with an individual measurement tolerance of ±1mm. Using algorithms described in our software section – section 5 – we are able to differentiate between range measurements representing the ground plane and range measurements representing obstacles.

The Point Grey Stereovision System uses images from three different cameras to find the disparity between similar pixels in each image. This disparity in pixel location results from parallax. This is used to create a disparity map of a single image. This disparity map has an inverse relationship with distance that can be used to calculate a point cloud of pixels. Each point has a location in three-dimensional space and a corresponding color value. Again, using algorithms outlined in our software section – section 5 – we are able to separate points which represent the ground plane from points which represent obstacles. Thus, we have two, redundant obstacle detection sensors. This design feature compensates for sensor imperfections to detect a higher range of object materials. Any material that does not reflect 905nm ultraviolet light cannot be detected by the LIDAR. Likewise, the Point Grey camera cannot generate a disparity map of highly uniform colors because it cannot differentiate distinct pixels. If only a single sensor was used to detect obstacles, our robot would fail either in cases of non-ultraviolet-reflective materials or uniformly colored materials. However, by introducing a redundancy, our robot is able to detect obstacles made of a much wider range of materials.

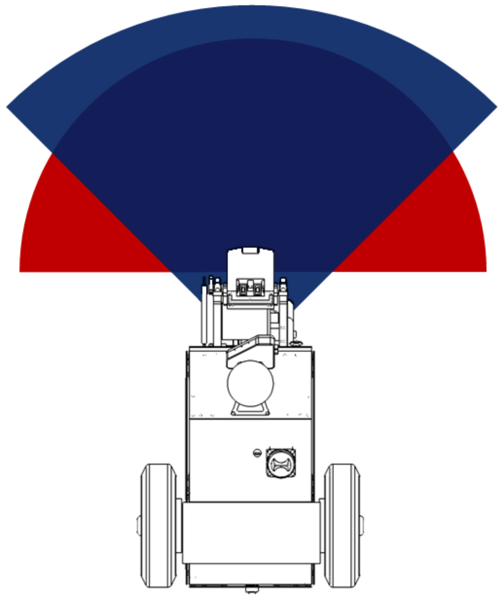
Figure 5: The red area shows the range of angles scanned by the LIDAR (180 degrees in total), while the navy section shows the area captured by the Point Grey when the front wheel is pointed forward. The Purple is the area scanned by both when the front wheel is pointed forward.

Our final sensor class is the line sensing class. To sense lines, we again use the Point Grey Digiclops Stereovision System. By removing obstacles found using the disparity maps, we are left with an image of only the ground plane. Using this data subset we find lines using the algorithms described in section 5. By only searching for lines in the ground plane, we are able to ignore line-like features on obstacles themselves. The team also attempted to introduce redundant line sensing by observing the energy of reflected LIDAR beams. Materials that absorb LIDAR's 905nm ultraviolet light more readily can be differentiated from materials that are highly reflective. Unfortunately, the energy of waves reflected of white painted grass cannot be differentiated from the energy of waves reflected of unpainted grass.

4.4 Computing

Computing onboard *Brian* is distributed between two computers. The first is a SmallPC brand rugged PC with a 2.4GHz Intel Pentium IV processor. The second is a Dell Latitude D620 laptop with a 2.16 GHz Intel Core II Duo Processor. These computers were selected due to their no cost availability. Both machines run Windows XP professional and are programmed using National Instruments' LabVIEW software. The two computers communicate via TCP/IP through a datasocket server hosted on the Dell. The Dell handles all image acquisition through a Rosewill RC-602 PCMCIA FireWire card. Images are processed on the laptop – as described in section 5 – in order to create an array of all points identified as lines and objects by the Point Grey System. These points are in Cartesian coordinates with the origin located in the camera's frame of reference. This array is then passed to the datasocket server. A separate machine is required for the Point Grey System because of the computationally intense nature of stereo vision image processing. Additionally, the laptop provides an easily stored and easily accessed onboard monitor for field testing and debugging.

The small PC reads from the datasocket server to incorporate vision data into cognition code. The SmallPC acquires and processes data from all other instruments onboard *Brian*. The PC utilizes three PCI slots to communicate a FASTCOM 422/2-ISA serial port, Sensoray 626 Analog & Digital I/O, and a Linksys LNE100TX Etherfast LAN Card. These cards are used to control motors, receive user button input, read encoder values, communicate with the LIDAR, read from the GPS, and access the Dell's datasocket server.

5 Software

Brian was programmed in LabVIEW for two major reasons. First, LabVIEW's extensive libraries made interfacing with hardware trivial. Also, from Professor David Barrett's prior experience of, we knew people without programming experience could learn the language in under a week. This was crucial, as only one member on the team entered with LabVIEW programming experience. Using LabVIEW allowed us to quickly learn a common programming language and rapidly deploy our design.

The software architecture used onboard *Brian* exploits the advantages of a reactive paradigm. The reactive paradigm of navigation differs from the canonical sense, plan, and act paradigm by removing the planning stage of the algorithm. Instead, the robot uses a simple algorithm to react to the environment around it. First we will describe the software *Brian* uses to sense the world in the following three subsections, then we will describe the algorithm that *Brian* uses to act, and finally we will discuss the advantages of a reactive paradigm for competing in the IGVC.

5.1 LIDAR Filtering

As described in section 4.2, the LIDAR returns measurements to the nearest obstacle for a 180 degree arc segment in front of the robot. This data must be processed to determine which data points represent obstacles and which points merely represent the ground plane. The 2009 IGVC rules stipulate that the ground will never exceed a fifteen percent grade. Our filter is based off assumptions made as a result of this rule. If we assume that the ground has a constant fifteen-percent grade incline in all directions – the maximum allowable incline – the result is a conical plane like the one illustrated in Figure 6. Any data point that lies above that plane must – as per the challenge rules – be an obstacle. We assume that all points below this plane are ground. With this algorithm, *Brian* can detect an obstacle with a height of six-inches or larger at a range beyond his forty-inch turning radius. This approach allows *Brian* to detect and pass obstacles with a height greater than six inches without slowing down. Because the head can oscillate, we can also detect an obstacle of a height of just over two inches at a range of nineteen inches in front of the robot. While this does not allow passing without stopping – as twenty inches is within *Brian's* turning radius – it does allow us to avoid collisions with even small obstacles. This filter has one primary advantage: speed. *Brian* can very quickly compare LIDAR data point with a pre-calculated value in a lookup table to determine whether or not the point is an obstacle.

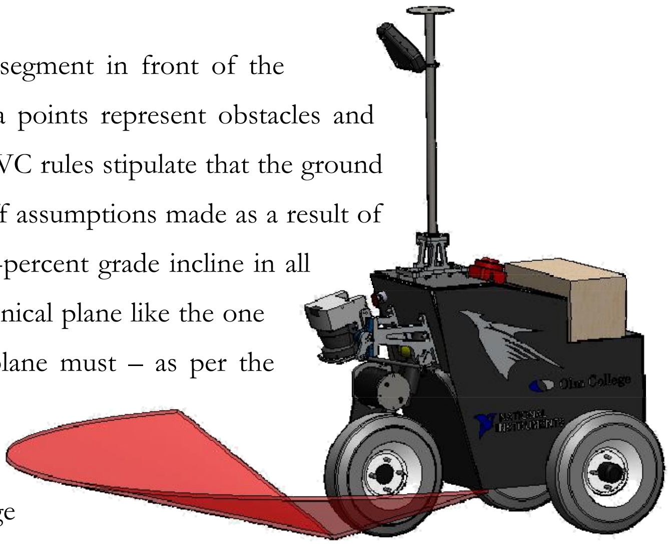
Figure 6: The red cone is a visualization of a 15% grade increasing in all directions around the robot. Brian assumes that any point reflected to the LIDAR above that plane is an object.

While this method allows us to detect obstacles, it does not help avoid potholes. To do this, the data filtered out as part of the ground plane can be processed. The data is first transformed into Cartesian coordinates in the LIDAR's frame of reference. Next the data points are processed in order. The altitude of each point is subtracted from the point that followed it in the scan. If the difference is found to be two inches ±0.5 inches, then that point is placed in a memory array. If the slope between a point and the point that follows it is negative, the point's location is placed into the array with a negative altitude, while pairs with a positive slope receive a positive altitude. When all points with proper slopes are placed into the array, an algorithm described by the following Turing Machine can be used to find which points in that array are part of a pothole.

1. Read altitude value of the first input point.
2. If it is positive, remove it from the array and go to Step 1, else continue.
3. If the second point in the array is negative, remove the first point and go to step one, else continue.
4. If the distance over the ground plane between first and second points is between 1.5 and 2.25 feet apart, both points are part of a pot hole. Remove them from the array and record them as part of a pothole.
5. If the memory array has one or fewer elements, terminate, else go to Step 1.

This ensures that a falling edge is followed by a rising edge. Also, it makes sure the distance between edges is at least one and a half feet and no more than two and a quarter feet – based on our knowledge of the challenge rules stipulating two foot potholes. We do not allow smaller diameter pothole cross sections to be detected because random noise generated by variance in grass height could then appear to be pothole. This will place two line segments in our occupancy grid, and not the pot hole's full circle. However, due to our obstacle avoidance algorithm's tendency to avoid obstacles with a large clearance, *Brian* will still avoid the entire pothole (the specifics of the algorithm will be discussed in section 5.4).

Figure 7: A disparity map of a level, flat ground plane (note some deviation in the ground is visible to the upper right as we could not find a completely flat patch of ground).

5.2 Vision Filtering

The goal of our vision filter is to find obstacles and lines on the ground. We accomplish this through two main filters: the first determines where the ground plane is, and the second determines what pixels on the ground plane correspond to the white or yellow lines.

The first filter is used to determine what the ground plane is and eliminate any other obstacles. As was mentioned earlier, the Point Grey camera returns a disparity value for every pixel that is inversely proportional to the distance from the camera. Because the camera maintains the same height and angle with the ground plane, the plane of the ground (excluding obstacles) should always have roughly the same disparity map. This flat map is shown in Figure 7. When obstacles, such as barrels, are added to the camera's field of view, the disparity map changes accordingly, as the obstacle locations have higher disparity values (as visible in Figure 9). A standard ground plane disparity map is subtracted from the current disparity map from the camera. After this subtraction occurs, the only remaining positive values are obstacles. Two filters are then applied. The first removes the ground plane and retains the obstacles, which are then added to the global obstacle certainty array. The second removes the obstacles and retains the ground plane. Some morphological post-processing is then done to clean up the image, and a ground plane mask is created. This mask is applied to the color image to return only the ground (Figure 8).

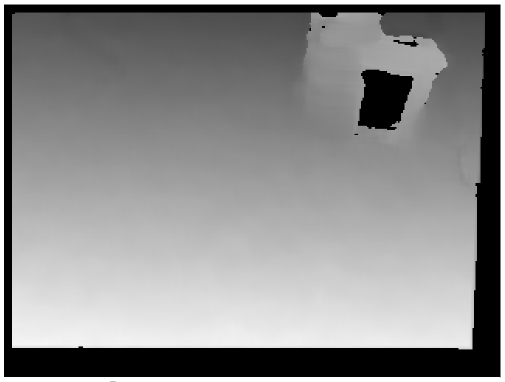
Figure 9: Disparity map showing some obstacle in the way.

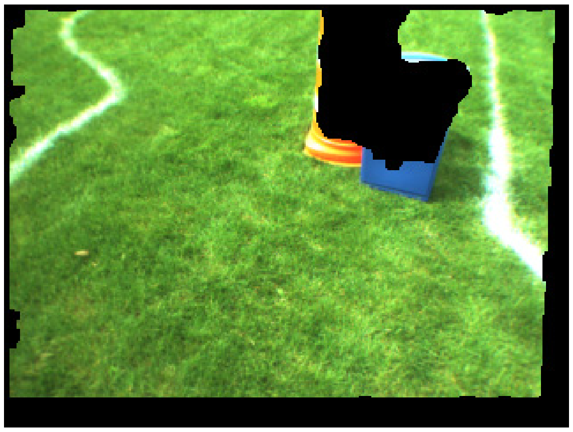
Figure 8: This figure shows the image after the obstacles have been filtered out. Note that while the filter is effective, it does miss the edges of the obstacles.

The second main filter is used to determine where the painted white lines are. We approached this filter using two different methods. The first was a simple blue filter. This was modestly effective because there is a large quantity of blue in white lines, but very little in green grass. Issues arose with this filter, however, because it required sitting with the robot looking at the lines and adapting the filter values whenever the lighting conditions changed, which is decidedly not robust. To create a more robust filter, we adopted a hue and lightness filter. Both of these filters are adaptive to the image, creating threshold values based off the average HSL values of the entire image and the standard deviation of each. This filter removes the most abundant color from the image, which makes the filter generally effective in a variety of different lighting conditions and on a multitude of different ground types (grass, asphalt, etc.). The results can be seen in Figure 10.

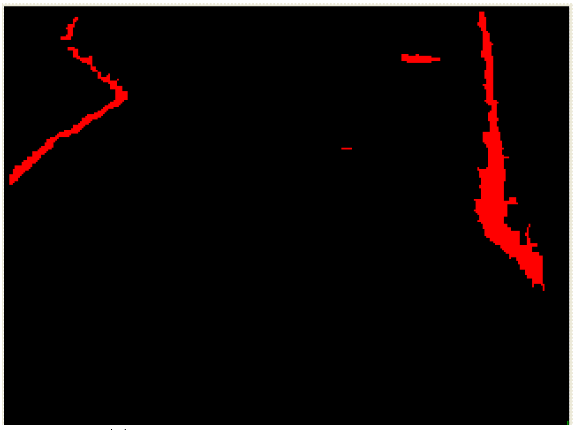
Figure 10: This figure shows the location of the lines in the image.

5.3 Mapping the Environment

In order to make sense of data received by instruments, it is placed into a global occupancy grid, represented by a two dimensional array. Each cell of this two-dimensional array corresponds to a three-by-three inch area of the race course. While three-dimensional data of each obstacle or line point is known by the robot, and used to detect the points placed in the occupancy grid, the altitude dimension can be disregarded in this case. This is a valid assumption because the race course never exceeds a fifteen-percent grade. Assuming a flat world in our data storage method introduces a maximum error of 1.3 inches at the ranges of zero to ten feet scanned by our sensors. Placing data into the occupancy grid can be done by subtracting the point's transformation into the global reference frame from the robots position in the global reference frame. Because this result is in inches, modularly dividing the coordinates by three determines the data's position in the occupancy grid (as shown in the following equation).

$$X_O = \mathrm{mod}_3(X_D - X_R)$$

$$Y_O = \mathrm{mod}_3(Y_D - Y_R)$$

Once the location within the global occupancy grid is determined, we increment the corresponding array cell. In our data representation, we differentiate between lines and objects by storing their data points in two different occupancy grids.

In order to keep track of obstacles and lines detected at previous cycles of computation, the occupancy grid arrays retain their values across cycles of computation. Instead of overwriting the value of cells with previously detected obstacles and lines, the value of the cells are incremented. Because our camera captures a 360x240 pixel image at the same 10Hz rate as our LIDAR – which only captures 180 pixels per scan – we must increment data for these two instruments differently. Any data point found by the LIDAR is incremented at a rate two orders of magnitude larger than the rate of increase for the camera. This creates a certainty value that is proportional to the magnitude of each cell in the array. However, over time, imperfect localization erodes the accuracy of data correlation over large distances. That is to say within the occupancy grid, the distance between currently sensed objects and objects sensed long ago is not an accurate correlation. As a result, in addition to having a memory of old previously sensed locations, our occupancy forgets over time. Every five computation cycles, we decrement each cell in the array the equivalent of one LIDAR data point increment. This means that over time, if an obstacle or line is not continually detected, *Brian* becomes less certain it exists, until he forgets it was ever there at all.

5.4 Reactive Algorithm

With occupancy grids described in section 5.1, the robot applies a simple algorithm to determine its action. Both lines and obstacles must be avoided. Thus for our reactive paradigm, we generate a third occupancy grid. This grid is the sum of both the line and obstacle arrays. Each cell in the resulting occupancy grid exerts a virtual force on the robot. This force is summed over all cells as defined in the following equation, where *F* is the total force, k is a tuned constant, c is the certainty value of the cell within the occupancy grid, d is the distance between the cell and the robot, and m and n are the number of rows and columns in the occupancy grid respectively.

$$F = \sum_{x=0}^{n}\left(\sum_{y=0}^{m} kc_{(x,y)} \frac{1}{d^2_{(x,y)}}\right)$$

The direction of this force determines the heading of the vehicle, while the magnitude of the resultant vector determines the robot speed. In order to decrease computation time, this algorithm is not actually applied to the entire occupancy grid. Instead, we apply the operation to a subset of the array. Due to the inverse squared relationship of distance, we know that any obstacle or line at a range of more than five feet exerts a negligible force on the robot. Thus, we only apply the calculation to a 41x41 element sub array, centered at the robot's current location.

5.5 Waypoint Generation

Clearly, force described in the previous section pushes our robot away from lines and obstacles. However, the robot needs to move to a destination, not just away from lines and obstacles. To achieve this, we add constant forces that push the robot towards our desired goal. In the navigation challenge, this is done by adding a single, constant magnitude force vector in the direction of the next GPS waypoint. This pulls the robot towards the waypoint, until it comes close to an obstacle or line. At that point, the force of the obstacle subsumes the force of the waypoint due to the shrinking magnitude of obstacle's the distance squared term. This repulsive force remains the primary determining factor in robot motion until the obstacle is cleared, the distance term grows, and the waypoint's attractive force subsumes the shrinking obstacle forces.

In order to direct the robot during the autonomous challenge, a sand piling method is used. The robot places a strong, constant repulsive force in the array cell the robot occupied three seconds ago. This helps propel the robot forward, and also helps the robot escape local minima. A local minima is a location on the course in which all the virtual forces push the robot toward a single point that is not the desired waypoint. After three seconds of being stuck at a minima, the robot has a force at its current location that pushes it out of the minima. The additional forces added to direct the robot are stored in an array separate from the occupancy grid as they are treated differently by the memory. These constant directive forces are added after calculating the sum of the occupancy grid forces.

5.5 Software Innovation

Brian's primary software innovation is his use of the reactive paradigm. The reactive approach offers a few key advantages. The first is speed. Based off of this year's rules, as well as investigation of past IGVC courses, the team knew that there would be a single closed path for the autonomous challenge. This means that advanced decisions about branching and avoiding dead ends would not have to be considered by our robot. This encourages a simple, robust solution to the problem. While there is only one path through the course, there are certainly more and less optimal driven courses through that path. Our algorithm pushes *Brian* to the center of the lane, instead of closely rounding corners on a more optimized course. However, in reading previous design reports, we quickly found the processing demands of computing optimal paths affected robots' top speeds. Finding the optimal path reduces the rate at which a robot can make decisions, and thus, lowers its top speed. This is most evident in our robot's course update rate, which occurs at 10Hz. This is about two times faster than traditional sense, plan, act algorithms – as compared to a robot with similar computing power, such as Princeton's *Kratos*. This allows our robot to travel a suboptimal path at the top allowable speed. This has two advantages; first, we were able to near double our top speed. While the suboptimal path may be longer, it is not twice as long. Additionally, the suboptimal path takes a wider berth around obstacles. This means there is a greater allowance for error in sensor readings and localization. Thus, by using a reactive paradigm we are able to both increase the rate at which we cover ground and decrease the probability of us running into obstacles and lines.

6 Predicted Performance of Vehicle:

6.1 Speed and Ramp Climbing

Our Doran Chimp base vehicle is a personal transportation unit designed to carry a 300lb rider at 11mph on a flat surface. The additional electric and mechanical systems on the vehicle are equivalent to a 180 pound person. Therefore, we are well within the specification of the original vehicle, and qualitatively expect a comparable performance. We have never had problems with the base vehicle being unable to carry our load. Our structural system even retains the original rider support, allowing for the sidetracked team member to ride the robot.

Quantitatively, the motor on the base vehicle is a 1.3 Horsepower 24V brushed DC motor, geared up by a 3:1 ratio. At full throttle with a rolling resistance of .065 (accepted value for a tire on grass), the theoretical maximum speed of the vehicle is 8mph for a flat surface and 2.5mph for a 15% uphill grade. In accordance with competition rules, we have hardware limited our main motor controller to 4.5 mph.

In general, we have kept our robot's speed software limited to 3mph for algorithm development, and have discovered that our algorithm efficiency is the critical speed limiting factor, rather than the restrictions of the base vehicle. A fast algorithm therefore directly correlates to a faster competition run speed, the primary impetus for our reactive algorithm paradigm.

6.2 Reaction time

Our cognition loop runs at 10Hz, meaning that our reaction time is limited more by the mechanical system than the software. While the computer is able to react to a new obstacle in 0.1 seconds, the steering system can theoretically slew 45° in 0.28 seconds, slowed by an estimated 5.2 Nm load on the steering motor (a Chairman AGM-12100T).

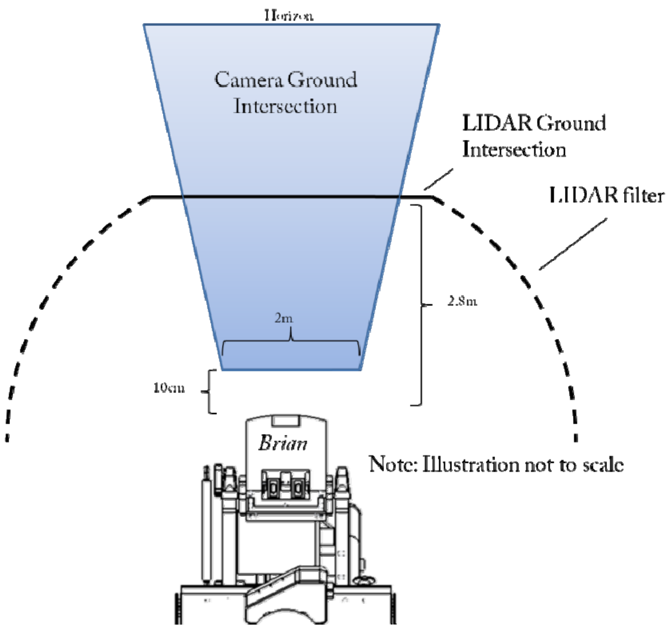
Horizon
Camera Ground Intersection
LIDAR Ground Intersection
LIDAR filter
2.8m
2m
10cm
Brian
Note: Illustration not to scale

Figure 11: Obstacle detection ranges for various sensors.

6.3 Battery life

The batteries installed on the robot are the original Group 27, 115 amp-hr., deep-cycle lead-acid batteries from the Chimp base vehicle. Doran Vehicles claims that a Chimp can travel for 30 miles on a single charge, which translates to 2.7 hours of continuous usage of the drive motor. We have worked on the robot for 10 hours of on-off usage on a full charge of battery. Ultimately, we have few worries about our battery capacity during competition.

6.4 Distance at which obstacles are detected

The obstacle detection distance is highly important, especially for a reactive system, because acting on an obstacle too soon or too late can cause unnecessary avoidance or collisions. Our LIDAR is mounted 75cm off the ground and tilted at a 15° angle, so our robot detects obstacles less than 2.8m in front of it. This may seem small, but with our 1.01m steering radius and 10Hz cognition loop, this detection distance provides more than enough time to avoid obstacles. However, one of our goals between now and competition is to increase our travel speed, so the LIDAR angle may change to adjust this reaction time.

Our Point Grey Digiclops stereoscopic camera is mounted at a 30° angle off the vertical, 1.8m from the ground, with an approximate 90° field of view. This gives us obstacle detection from the horizon line to 10cm in front of the robot.

6.5 Autonomous Performance

Our algorithms for both the autonomous and navigation courses use the same obstacle avoidance method to guide the robot. Our autonomous algorithm for switchbacks and islands is a "sand piling" method, where the robot uses its position a few seconds before to guide the direction it needs to go, avoiding obstacles and lines along the way. By tuning that time parameter, we can optimize the robot for different situations. For example, using a previous position from a long time ago will guide the robot down straight path with many obstacles, rather than a winding path with fewer obstacles. We will be using our on-site testing runs to tune this parameter for the competition course. Testing has proven this method to be rather robust for switchbacks and islands; however, it performs poorly on long dead ends.

6.6 Navigation Performance

Our uBlox GPS is our most troublesome (and cheapest) sensor. The current GPS performance is a 2m radius watch circle, which is just barely enough to navigate the valley waypoints effectively. Accordingly, we do not use the GPS for localization, but instead only update the waypoint locations in the robot's internal map. Our obstacle avoidance algorithm is extremely robust given static waypoints; therefore, we predict efficient travel between waypoints but only 2m accuracy upon arrival.

7 Conclusion:

This paper outlines the design and implementation of Brian, a fully autonomous vehicle created for the 2009 Intelligent Ground Vehicles Competition. Five undergraduates at the Franklin W. Olin College of Engineering made *Brian* in four months. In the process we learned about engineering design and gained significant problem-solving experience through the realization of a complex, interdisciplinary project. We would like to thank National Instruments and Olin College for their support, as well as Professor David Barrett for his guidance. We believe that Brian represents an innovative design in the IGVC and hope to be competitive in the 2009 competition.