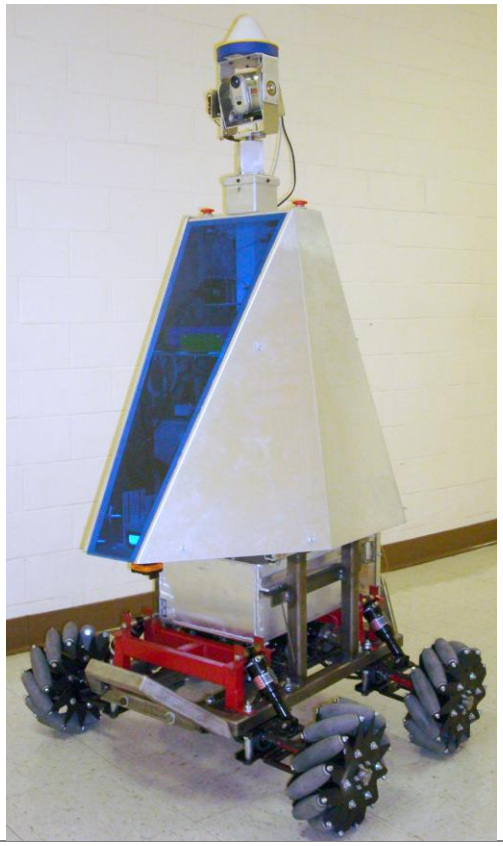

Archon

IGVC Design Report

Justin Milam, Matthew Duncan, Caleb Tote, Jeremiah Weakley, and Andrew Scott

I, Dr. Robert Riggins, Professor of the Department of Electrical Engineering Technology at Bluefield State College, do hereby certify that the engineering design of Archon has been significant and each team member has earned at least two semester hours credit for their work on this project.

Signed, Date

_____

Phone: (304) 327-4134 E-mail: briggins@bluefieldstate.edu

Contents

1. Introduction

1.1 Overview

The Bluefield State College (BSC) robotics team is proud to present Archon for entry in the 18th Annual Intelligent Ground Vehicle Competition (IGVC). The Bluefield State team has designed Archon from the ground up with many innovative changes in both hardware (Sections 3.4/3.5), and software (Section 4.9). Archon is a culmination of all of our knowledge and experience from ten years of attending IGVC. By analyzing past IGVC performances, we have strengthened every aspect of the robot's design. We believe Archon will have a strong showing at this year's competition.

1.2 Team Overview

BSC's robotics team is comprised of undergraduate students from the Electrical Engineering Technology, Computer Science, and Mechanical Engineering Technology departments. The robotics team is organized in a hierarchal structure with a team lead overseeing the entire project. The team is then divided into both hardware and software divisions. There is a leader in each of the respective divisions who further dictates what is to be done by whom. The hierarchal design can be seen in Figure 1. The hardware team's responsibilities include the physical design, electrical layout, and fabrication. Archon's hardware is further discussed in Section 3. The software team is responsible for the layout of the applications, integration of sensor information, and navigational algorithms. The software for Archon is discussed in more detail in Section 4.

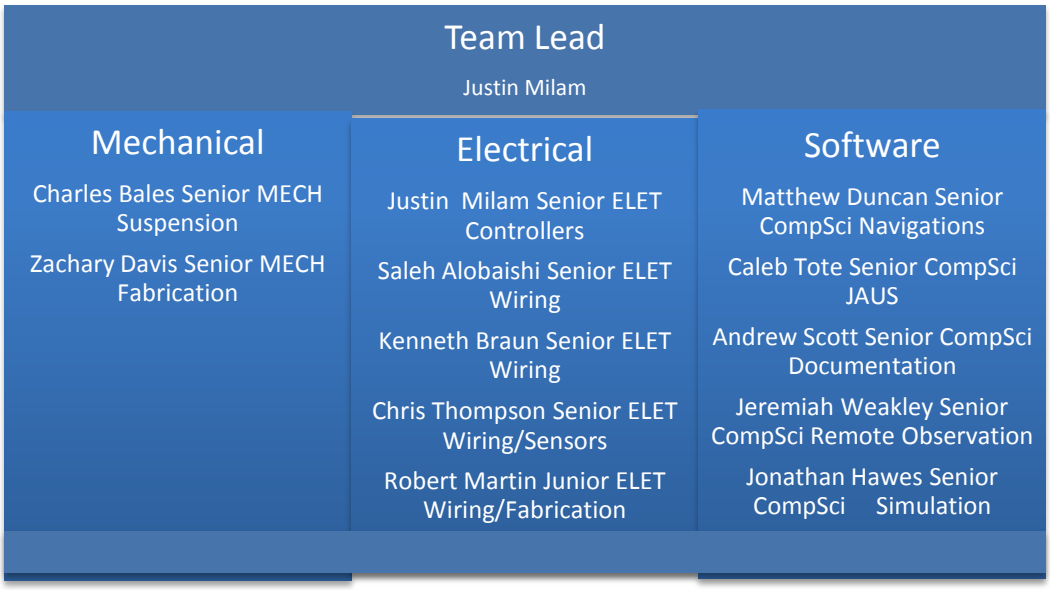

Figure 1: Team Hierarchy

2. Design Process

2.1 Design Methodology

The design methodology that was used by the robotics team is outlined in Figure 2. The process used began with a full analysis of the previous year's IGVC competition. The robotics team used this post-IGVC analysis to scrutinize every event that happened with the robot while at the IGVC. Each event was then evaluated on whether it met our expectations and whether design or application could be improved upon. This analysis was then documented and, following the team hierarchy described in Figure 2, given to their respective groups.

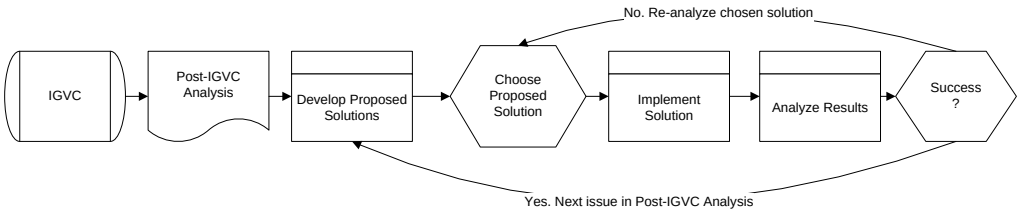

Figure 2: Design Methodology

Once the groups receive the topics they are to review, they follow the design process flow specified in Figure 2. The team meets to develop proposed solutions for the problem they are improving. Then each proposed solution is analyzed to find which has the greatest chance for success. This solution is then implemented using the outline described in the proposed solution phase of Figure 2. The results are then carefully analyzed to determine if it fixed the original problem. If the results are satisfactory, then the implementation is declared a success and the team moves on to the next problem. If the proposed solution did not rectify the problem, then the team reverts back to the development stage and the steps are followed again until a solution is labeled a success.

2.2 Design Objectives

Prior to the development of Archon, there were several design objectives that we felt needed to be implemented to give the robot a higher performance level. These objectives are classified into two different categories: software and hardware. Some of these objectives are things that we wanted to see in Archon, while others are requirements and rules of IGVC.

Hardware:

1. Omni-directional capable vehicle
2. Rugged/Durable frame
3. Safety
4. Modularity
5. Performance requirements specified by IGVC:
    - Size requirements
    - 20lb payload
    - 15% incline
    - 5mph speed limit

Software:

5. Hardware independent software
6. Seamless JAUS integration
7. Safety

3. Hardware Design

Archon was completely designed by the students at BSC, including the physical platform and electronics. During the design phase, the team sought to address as many limitations of the previous robots as possible. Using our accumulated knowledge from all our earlier robots lead to innovation in every aspect of the mechanical and electrical systems.

3.1 Mechanical Design

This section describes the physical characteristics of the robot. Archon is divided into two main parts: top and bottom. We chose this type of design in order to make our robot modular. This modularity allows Archon a great deal of versatility. For instance, depending on the terrain, Archon can have different drive systems installed such as four-wheel skid-steer, mecanum drive, omni-wheel drive, and tank-tread operation. The same top, which houses most of the electronics and all sensors, can be used with these different bottoms. For IGVC 2010, Archon will use the mecanum drive. In the same way, different tops could also be placed on each bottom. The team made the cables and connections between top and bottom to be as simple and standard as possible.

3.1.1 Drive system

When designing Archon, we wanted to make a robot that was capable of true omni-directional movement. In order to achieve this, we decided to use mecanum wheels. Mecanum wheels allow a vehicle to move in any direction and in any configuration. In other words, the robot can move in any translation (2D movement without rotation) as well as add any rotation while in transition. Mecanum wheels operate on the principle of utilizing opposing force vectors. Each wheel has one main wheel with several free rolling wheels mounted around its periphery. These rollers are mounted at 45 degree angles which cast their force vectors in different directions. By spinning each wheel in a different way and with varied speed, the vehicle can achieve omni-directional movement. We use four motors in our vehicle for several reasons. First, mecanum wheels must be driven to function properly. This means that the wheels cannot be free spinning in order to operate effectively. Second, by using four motors, we are able to carry a heavier payload and move significantly faster.

The four motors are fastened to the steel tubing, one motor per wheel, producing an independently-driven four-wheel-drive system. The stall torque of each motor is 343.27 ounce-inches. IGVC requires that the robot must climb at least a 15% incline. Each wheel has a radius of five inches and supports a static weight of 62.5 pounds. The required torque,T_R, for each motor to maintain constant speed going up a 15% incline is calculated as follows:

$$T_R = 62.5lbs * \sin\left(\frac{15}{100} * 90°\right) * 5\ inches * 16\ ounces\ per\ lb = 1167\ ounce - inches$$

The team used 27:1 gear reducers on each motor, thereby increasing the stall torque, T_M of each motor to:

$$T_M = 343.27\ ounce - inch * 27 = 9268.3\ ounce - inch$$

Since $T_M \gg T_R$ Archon will have no problem navigating up or down a 15% incline.

Archon's suspension system was designed and implemented by our mechanical engineering members. In order for a mecanum-wheeled vehicle (or any skid-steer vehicle) to operate effectively and efficiently, all of its wheels need to be in

constant contact with the ground. In a mecanum wheeled chassis, if one wheel leaves the ground, it will drive somewhat erratically. Therefore, Archon is designed to maintain ground contact with up to two inches of ground height differences on each wheel (4 inches per axle.)

3.1.2 Top

Archon's top houses all of the sensors and the computer. The sensors are mounted in such a way as to allow for the greatest visibility for the robot. With the current configuration, Archon is capable or "seeing" 360 degrees. There are two 270 degree LMS units mounted on Archon on opposing sides. These two devices overlap and allow for no blind spots around Archon. The camera is mounted on an extendable and positional shaft. This head, allows us to adjust the horizontal and vertical aim of the camera.

Perhaps Archon's greatest innovation is the fact that he is completely omni-directional. This means that not only is the robot able to move in any direction, there is no difference between front and rear. Archon is completely symmetrical which allows it to make a 180-degree change in direction instantaneously.

Archon's structure on top is all lightweight aluminum in order to keep the center of gravity low. The weight of the top, including all items such as sensors, is 50 pounds.

3.1.3 Bottom

The main frame of Archon's bottom half is made of $1/8^{th}$ inch steel tubing. This allows the frame to support the weight of the robot as well as the payload specified by IGVC rules. The added weight incurred by using steel keeps the center of gravity low on the robot to prevent tipping. The total weight of the bottom, including wheels and battery, is 200 pounds.

Each wheel also has independent suspension. The team designed and fabricated this suspension system in order to accommodate our choice of wheels. As stated above, one of the properties of mecanum wheels is that when one wheel loses traction unpredictable movement will occur. The suspension system was designed and implemented so that even if Archon hits a pot hole in the road, all four wheels will remain in contact and keep traction. It was implemented using dual control arms, and is slip-shaft driven to achieve an independent suspension at each wheel. Each shock is able to be adjusted from 0 to 500 pounds of pressure. This range far exceeds Archon's 250-pound total weight. Due to Archon's symmetry in structure as well as weight distribution, each wheel has a static weight of 62.5 pounds. The suspension system is completely adjustable in both rigidity and rebound speed.

3.1.4 Materials

Archon is constructed from a mix of materials to fit each unique piece. The control module at the top as well as the camera mast is made from aluminum. Using a light yet rigid and durable material like aluminum is perfect for providing a protective shell around Archon's more intricate parts. As aluminum is non-magnetic, it also lowers any chance of the frame interfering with electromagnetic sensors such as the compass. It acts as a great heat reflector in case the outside temperature is high during IGVC. The bottom, as mentioned before is made of steel tubing. Steel is able to take the harder strains placed on the frame by the motors as well as the added weight of the suspension, battery, and drive system.

3.2 Electrical Design

We've taken many of the electrical designs tested in our previous robots and used them to their fullest potential in the Archon system. The new compact size of Archon's body offered interesting electrical challenges during the design process, but this has led to a very ergonomic and functional design.

3.2.1 Power

Archon's frame is the smallest we've ever created. This means we do not have the same amount of space available for a large battery system. Such a problem led to the need for a new way to look at our power source. We replaced the large two battery systems of previous BSC robots with a single twelve-volt battery. An upgrade has also taken place in the type of battery we use. In the past, we have made use of the same kind of batteries you would find in any car. Now we use a single commercial grade battery (rated for 75 amp-hours), more often found in large trucks. This allows for an extended battery life.

All devices on the robot including the four motors are powered with a 12-volt industrial-sized battery (also located in the bottom.) This battery is rated for 75 amp-hours of total energy. The peak efficiency current of each motor is 19.8 amps. Assuming the average running current of each motor is about one-half of the peak efficiency current, then the motors would typically require 40 amps during a run. All other devices (GPS, both LMSs, microcontrollers, compass, wireless devices) take less than five amps in full operation. Assuming a battery efficiency of 65% we calculate a predicted time between charges (T_C) as follows:

$$T_C = 75\ ampHours * 0.65 * \frac{1}{45}\ amps = 1.08\ hours$$

Evolving to a new battery type is a definite improvement, but this still is less than the total amp hours we had available in older generation BSC robots which used a two-battery system. To compensate for having less battery life, we designed the frame such that the single battery can be removed easily and quickly to replace it with a fresh battery. A battery rail system was developed for Archon to make this battery replacement process easy. The battery, while protected in an aluminum box, can now be removed quickly from the wheel base and replaced with another.

3.2.2 Sensors

The following is a list of Archon's sensors and a brief description of their functions:

Device:	Function:
CSI Wireless DGPS Receiver and Antenna	Retrieves latitude and longitude.
Two Hokuyu 270-degree LMSs	Sweeps 270 degrees for object detection
Maretron Solid State Compass	Determines Archon's heading
Sony HandyCam Camcorder	Readily available vision system

Archon's sensors are placed in strategic positions around the robot so that they provide a full field of vision. A 270° Laser Measurement System (LMS) is placed on each end of Archon, giving the robot complete LMS coverage. The camera direction can now move to point anywhere on the 360° horizontal circle, giving vision coverage all around the robot. This suite of sensors allows for Archon to have complete sensor coverage of the environment.

3.2.3 Computer Systems

Archon's computer is a Dell Vostro (15" laptop). It is an Intel Core i5 based machine with 4 GB of RAM, running at 2.27 GHz. It runs on Windows 7 Professional. Archon communicates with its external devices using USB to RS232 adapters and USB2.0. This computer specification gives us power that we can use to make quick control decisions.

3.2.4 Actuators

With the modularity of Archon, we can use a large range of different actuators. Our current configuration make's use of four twelve-volt DC motors operating in a skid-steer setup. We have calculated that this motor design allows the robot to perform at five miles per hour. These calculations are as follows. Each motor has a no-load rpm of 5310 rpm, and a gear reduction of 27:1 giving each motor a no-load rpm of 196.7 rpm, or 3.28 revolutions per second. The wheels have a diameter of 10 inches and we assume a true rpm of 80% of no-load rpm. Therefore our calculated maximum speed S_M is given by the following:

$$S_M = 3.28\,rps * 10 * \pi\,inches * 0.8 * \frac{1}{12} feet\ per\ inch * \frac{3600}{5280} = 4.68\,miles\ per\ hour$$

3.2.5 Safety

Safety is always a major concern with our robotic systems and Archon is no exception. With both hardware and software methods of disabling the robot, we believe Archon is an incredibly safe machine. A twisting switch, referred to as a hard E-Stop, is the most definitive way of shutting down the system by manually cutting the batteries from the system. There are two easily accessible manual pushbuttons called the soft E-Stops which cut communication from the control system to the drive control, effectively stopping the robot while retaining power to the system. A radio-controlled (RC) wireless E-stop is

also incorporated into the control system to allow an operator to stop the robot from a long distance. Large numbers of fuses and breakers have also been placed to prevent any harm to equipment or team members.

3.4 Hardware Innovations

Archon's suspension system and mecanum wheels are amazing innovations in autonomous robots. Our suspension system gives our robot an extra degree of proficiency in both durability and ruggedness. It takes a lot of the abuse off of the electronic components and allows for our robot to go places that are normally unreachable due to impassible terrain. Archon's mecanum wheels are also very innovative. In addition to the type of terrain that Archon is capable of traversing, it can now traverse it in any way that is necessary. The mecanum wheels reduce the risk that Archon will get in a situation that it cannot get out of. Also, the robot is more efficient in maneuvering with very little time being wasted on extravagant maneuvers that are innate in normal skid steer vehicles.

3.5 Electrical Innovations

Archon's electrical systems have been revamped over previous systems designed by BSC. This year, we have implemented our own custom motor controller (an eight-core Propeller microcontroller from Parallax) that allows us to have great control over how Archon navigates through its environment. We have implemented a way of turning devices on and off via our software interface as well. This is done through a serial connection to a microcontroller (Microchip PIC18F452.) This design will allow Archon to be more power efficient in that when a certain device is not needed, it can be turned off and then back on again by the software or user. This microcontroller will also give the software feedback about sensors. This is important in the event of a hardware malfunction. The microcontroller could easily detect irregularities in device operation and report them to the PC, allowing the PC to make a decision on whether to ignore faulty sensor data.

4. Software Strategy

4.1 Software Overview

The BSC team's autonomous navigation algorithm is a six-step loop including map generation, sensor integration, map augmentation, goal selection, path creation, and control decision. The algorithm goes through each of these steps once per loop in the order shown above. This section will explain each step of the team's autonomous navigation algorithm in detail by using a data visualization example. This example is loosely based on the real world example of Archon ona practice course at BSC.

4.2 Map Generation

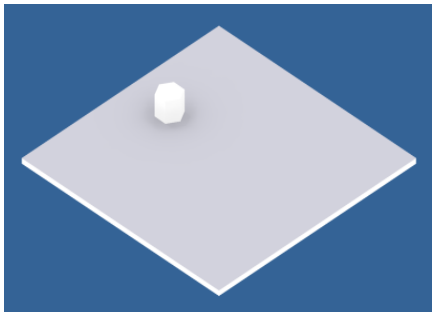

Figure 4.1: The Initial State of the map

The first step, called "map generation," allocates a two-dimensional array in RAM called the "map." The map stores values that will indicate obstacles, markings, and the path. The size of this map depends on the desired resolution and sensor field of view. For IGVC the team uses an 80 by 80 node map. Each map node represents four square inches of real world space because four inches is the size of the smallest obstacle on the IGVC course. Coordinates for the map start in the upper left corner (0, 0) and end in the bottom right corner (79, 79).

The robot is located at coordinates (40, 60) where there are 60 nodes (20.0 feet) of space in front of it, 40 nodes (13.3 feet) to each side, and 20 nodes (6.7 feet) to the rear. When the algorithm initializes the map, all nodes are assigned a value of 1000. At the beginning of the loop, one can think of the map as a flat field with a constant altitude of 1000 units surrounding the robot as shown in Figure 4.1. The data visualization example throughout the rest of this paper uses a smaller, scaled down map (12 by 12) in order to save space. Throughout this example, the robot will be indicated by a hexagon.

4.3 Signal Processing

Sensor data, such as compass, camera, LMS, etc., is brought in from the sensors and placed on the map. The format of the data from each sensor is very different from all others, and therefore the algorithm must convert all the sensor data to a common geometry so that the data can be located on the same map. First, this paper will discuss the different formats of the data.

The LMS returns data in the form of an array containing 181 elements. Each element represents a vector corresponding to the distance from the robot to an object at that angle. The algorithm uses the vector to determine the appropriate node and to assign a value to that node indicating an LMS obstacle. The value assigned to LMS nodes is higher than the value assigned to any other obstacle/marking node because the LMS is the most accurate sensor on Archon.

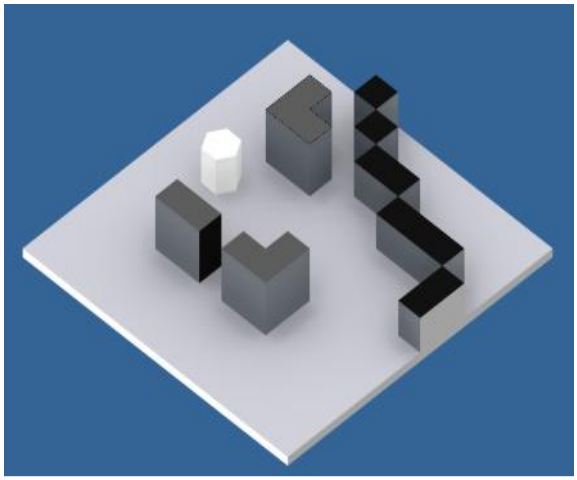

Figure 4.2: Sensor data representing object detection.

The camera returns data in the form of a two-dimensional picture. The image the camera gives the software is distorted both radian and geometrically. The algorithm processes this distorted image and converts it to a 181-element array similar to the LMS array. As with the LMS, the array elements represent a vector corresponding to the distance from the robot to an obstacle/marking at that angle. The algorithm then determines the appropriate node and assigns a value to that node indicating the camera obstacle/marking. In the example, the LMS nodes are distinguished from the camera nodes by height, with LMS nodes being higher than camera nodes.

4.4 Map Augmentation

	F	F	F	F	F		
	F	F	F	F	F	F	
	F	F	**L**	F	F	F	
	F	F	F	**C**	F	F	
	F	F	F	F	F	F	
		F	F	F	F	F	

Figure 4.3: Two fat layers surrounding two obstacle nodes.

The third step of the algorithm, called map augmentation, prepares the map for the path planning process. In order to save the path planning process execution time, the software determines the available space that the robot is capable of passing through. To do this, the algorithm creates "fat layers" around each obstacle/marking that is placed on the map in the previous step. A fat layer is simply a group of fat nodes that surround an obstacle/marking. A fat node has a special value lower than a camera or LMS value but higher than the normal values on the map. When surrounding a node with a fat layer, only clear nodes will be assigned a fat value. Nodes that have already been given another value are not assigned a fat value. Figure 4.3 shows what two fat layers look like on a map. In the figure, the F's represent fat nodes, the L represents an LMS obstacle, and the C represents a camera marking.

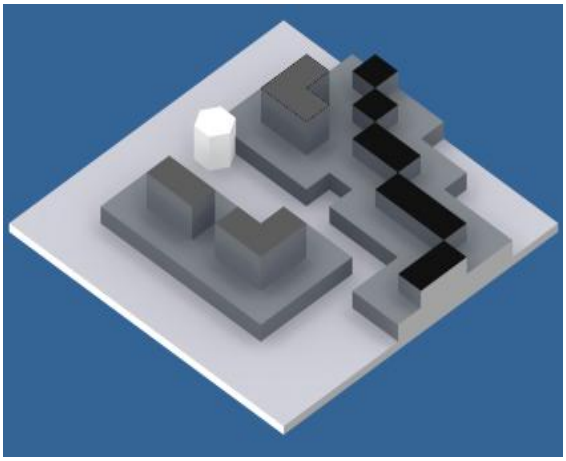

Figure 4.4: Fat layer added to the example map

By surrounding each obstacle with a fat layer, the software restricts how close the path planning process will allow the robot to pass to the obstacle/marking. For every additional fat layer, the algorithm prevents the robot from passing four more inches closer to the obstacle/marking. So for four fat layers, the path planner cannot get closer than 16 inches away from the obstacle/marking. This is necessary because the robot is represented as a single node on the map. (Each node represents four inches by four inches of real space.) In reality, however, the robot is 26 inches wide. The team uses four fat layers for IGVC because 16 inches is slightly larger than half the width of Archon. Figure 4.4 shows what the map looks like after one fat layer is augmented to the obstacles and markings that were added in the previous step.

4.5 Waypoint Navigation

In the fourth step, the algorithm must select a desired destination on the map called the "goal node." After this step, the goal node will have the lowest value of all nodes on the map. Nodes that contain an LMS, camera, or fat value cannot be the goal node. Also, the goal node must be reachable from the robot's position. For it to meet these conditions, each clear node on the map is considered a possible goal node, and each is entered into a special weight equation shown in Equation 1. Elements in this equation are of two types: goal node characteristics (d, α, β, S, G) and parameters (P_d, P_α, P_β, P_S, P_G). Goal node characteristics are attributes that are important in order for the possible goal node to be the final selected goal node. Each characteristic is defined in Figure 4.5.

$$Weight = (d \times P_d) + (\alpha \times P_\alpha) + (\beta \times P_\beta) + (S \times P_S) + (G \times P_G)$$

Equation 1: Weight Equation

Characteristic	Definition
d	A measure in meters of the distance between the robot and the possible goal node
$\propto$	A measure in degrees of how in line a possible goal node is with the actual direction of the GPS waypoint with respect to the robot
β	A measure in degrees of how in line a possible goal node is with the straight ahead direction of the robot
S	A measure of how in line the node is with the slant of the map (Slant is discussed below)
G	A measure of the gap on either side of the possible goal node if obstacles are present. (Gap is discussed below.)

Figure 4.5: Definitions of goal node selection characteristics.

The slant characteristic is the direction that objects on the map tend to follow. The best way to understand slant is it to think about how a person driving a car knows which way a road turns. While driving, one can get an idea which way the road turns by looking at the edges of the road where there are lines, tree-lines, buildings and other such markings that run more or less parallel with the road. By noticing the trend (or slant) that these markings and objects follow, one can anticipate which way the path will go.

The gap characteristic measures the amount of space to the next obstacle node that is on either side of the possible goal node. If there aren't any obstacle nodes present on one or both sides, then this characteristic is assigned the value zero. Gap specifically looks for ten feet gaps because the path width of IGVC is approximately ten feet. If the gap exceeds ten feet, then the gap characteristic is assigned a lower value. In this way, the gap characteristic favors ten feet wide paths.

Parameters are user-defined values and represent the importance of these characteristics for finding the goal node. The value of each parameter is set by a human controller prior to run-time. An operator can change these parameters if he feels that a certain characteristic will be more important to the current run than other parameters. Next, the process of how nodes are entered into this equation will be described.

The process of selecting a goal node starts at the robot's position on the map and iterates through all possible goal nodes. The algorithm checks the next row above the robot to find the left and right boundaries of that row. These boundaries can be either obstacles or the edge of the map. The first non-obstacle node is made the initial candidate goal node, and then the software checks the next possible goal node. The algorithm passes the clear nodes in the row through the weight equation (Equation 1) until it finds the boundaries. Each time a node has a higher weight than the current candidate goal node, it is made the new candidate goal node until all possible goal nodes in that row have been passed through the equation, or until the software finds the boundaries. The algorithm then successively moves to the next higher row until that row is completely blocked or the top of the map is reached. The path is considered blocked if every node across the map contains a value that represents an obstacle, marking, or fat node.

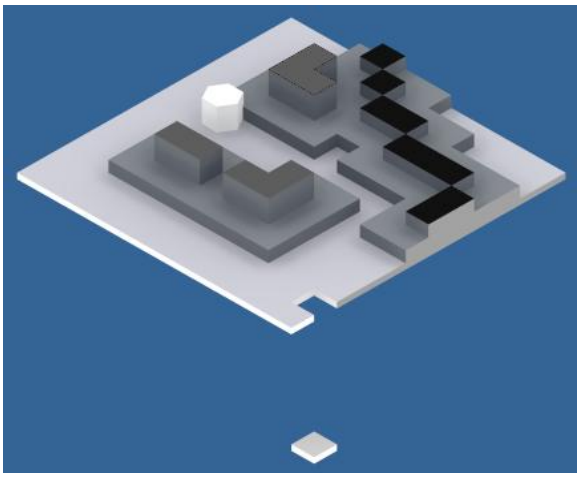

Figure 4.6: Goal node selected

Once the end of the path is found, this process goes back and checks any other existing paths. Thus, only nodes that are reachable from the robot node are considered possible goal nodes. After all the possible goal nodes have been passed through the weight equation, the candidate goal node with the highest weight is made the final goal node. On the map, the chosen goal node is assigned a value of zero. Figure 4.6 shows the result of goal node selection and shows the relationship between the goal node and the rest of the map (the value of the goal node is zero while the value of the flat plane is 1000). Next the software must find the shortest path to the goal node from the robot.

4.6 Path Creation

The fifth step, path creation, finds the shortest path from the robot to the goal node. Since the fourth step assures that the goal node is always reachable, there is always at least one path between robot and goal. The path creation step contains three sub-processes: the ripple algorithm, the waterfall algorithm, and the smoothing algorithm.

The ripple algorithm is the first sub-process in this step. Its job is to assign increasing values to clear nodes between the goal node and the robot. This process starts at the goal node which was assigned a value of zero in the previous step. It then assigns a value of one to all the clear nodes surrounding the goal node. Each clear node that touches a node that was given a value by the ripple algorithm is then given the next higher integer value. Nodes that represent obstacles, markings, or fat layers are skipped in this process, which continues until the robot is reached. Once the robot is reached, it is given the next value in the sequence, and the process stops.

Many paths from the robot to the goal node may exist, but one is the shortest. Figure 4.7 shows what the map for the example looks like after assigning increasing values to the nodes between the goal and the robot. In this figure, F represents a fat node, L represents an LMS object, C represents a camera marking, R represents the robot, G represents the goal node, and the numbers 1 -11 represent the values that the ripple algorithm assigns. For the data visualization example, Figure 4.8 shows how values increase from the goal node to the robot. Next, the software must determine the shortest path from the robot to the goal node.

		F	F	C	C	F	**3**	**2**	**1**	**G**	**1**
		F	C	F	F	F	**3**	**2**	**1**	**1**	**1**
		F	C	F	**5**	**4**	**3**	**2**	**2**	**2**	**2**
	F	F	C	F	**5**	F	F	F	F	**3**	**3**
	F	C	F	F	**6**	F	L	L	F	**4**	**4**
F	F	C	F	**7**	**7**	F	F	L	F	**5**	**5**
F	C	F	F	F	**8**	**8**	F	F	F	**6**	**6**
C	F	F	L	F	**9**	**9**	F	L	F	**7**	**7**
F	F	L	L	F	**10**	**10**	F	L	F	**8**	**8**
	F	F	F	F	**R**	**11**	F	F	F	**9**	**9**
								11	**10**	**10**	**10**
								11	**11**	**11**	**11**

Figure 4.7: Assignment of numbers in the ripple algorithm.

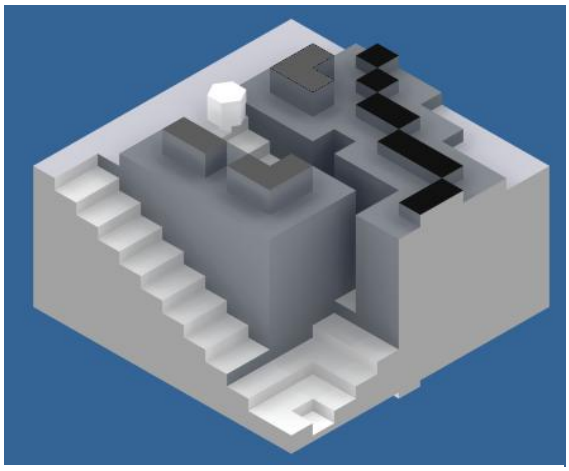

Figure 4.8: Results of the Ripple Process

To determine the shortest path, the software uses what we call "the waterfall algorithm." The waterfall algorithm starts at the robot and works its way "downhill" to the goal node. In the previous step, the ripple algorithm assigned the robot an integer value, which is the starting point for the waterfall algorithm. Starting at the robot, the algorithm looks for the next sequentially lower integer value in the path. Since there may be multiple options, the algorithm checks each choice's alignment with the goal node. It will choose the node that is most in line with the goal node as the next hop on the path. This process continues until a path from the robot to the goal node has been created.

Due to the nature of the map, each change in direction will be either 45 or 90 degrees. So the calculated path is the shortest one from the robot to the goal node constrained by this limitation. However, it is sometimes possible for the algorithm to shorten this path even more, by using the smoothing process.

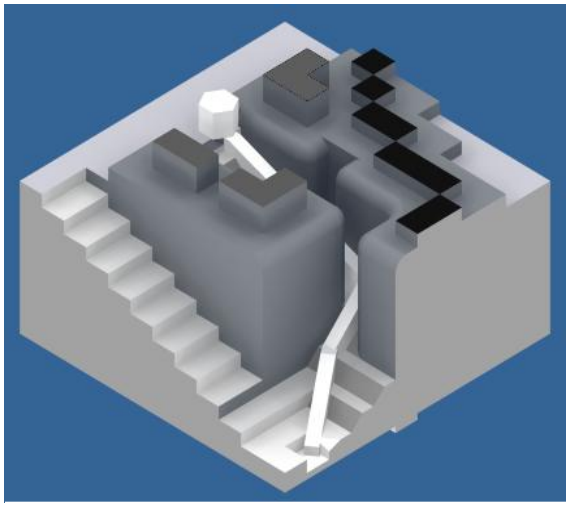

Figure 4.9: The final path in the example.

The smoothing algorithm takes the path that is created in the previous sub-process and "cuts the corners" wherever possible. For example, instead of going up a number of nodes and then taking a sharp turn, the smoothing algorithm will change the path to make it go straight to the most distant node along the path as long as no obstacle/marking/fat nodes exist in between. The smoothing algorithm will only cut these corners if there is no obstacle in the way. At the end of this sub-process the path to the goal node uses as many smooth turns as possible. Figure 4.9 shows the final path from the robot to the goal node after it has been smoothed.

4.7 Control Decision

The final step in the autonomous navigation algorithm is the control action step. In this step, the software tells the hardware what to do based on the path-finding decisions made in the previous steps. For this step, the software must send speed and direction commands to the motor controller. Speed is calculated based on how close the robot is to any objects. If the robot is close to many obstacles, it will not go as fast so that it can make tighter turns. If there are no obstacles around, then it will choose to go as fast as possible toward the goal. The software uses the initial segment of the planned path (starting from the robot) to calculate direction. Once speed and direction have been calculated, the computer sends the information to the motor controller which in turn sends the correct voltages to the motors.

4.8 JAUS

This year, the Bluefield State robotics program will be competing in the JAUS competition during IGVC. We feel that our approach to implementing network communications between robotic platforms is unique and efficient. Through the combination of pre-compiled JAUS specific C++ libraries and parallel processing, we are confident that our design will perform successfully.

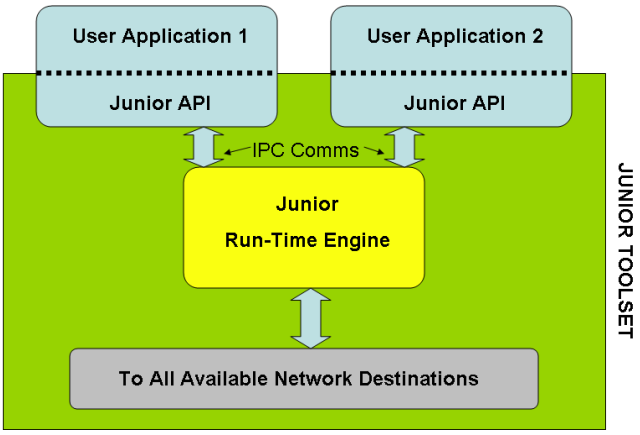

Figure 4.10: Junior Toolset

The coding strategy for JAUS is sophisticated since it relies heavily on high-level functions and capabilities of the C++ language. JAUS_NEIGHBOR is a user defined class that is created to tend to new connections. The motivation for designing a class structure for JAUS comes from the high demands and the necessary tools that are included with the JAUS architecture. A class orientation seemed to be the most logical design. We have encapsulated the JAUS_NEIGHBOR class as a "friend" member within our primary Robot class. This serves a few purposes: (a) this design allows the JAUS protocol to have access to necessary data members within the Robot class, while still keeping these variables private and protected from exterior objects and functions – (b) implementing JAUS_NEIGHBOR as a friend class also ensures a clean and efficient coding implementation, potentially cutting down time spent on troubleshooting.

To assist in the implementation of JAUS, we are coordinating with an open source JAUS specific C++ set of libraries, called *Jr Middleware*™. These prebuilt libraries allow us to ensure accurate connections with a JAUS device, while also enabling our team to focus more time on tuning how the robot responds to these JAUS commands. Our implementation of *Jr middleware*™ enables preassembled tools to act as a "middle-man" between our robot and the JAUS device. This can be seen in Figure 4.10.

The Bluefield State implementation of JAUS is unique in another way, however. With the advanced technology of multithreading, BSC has designed an implementation of JAUS that will allow interoperability between autonomy and JAUS control. As our application enters into "navigate mode", we instantly start a separate thread of execution which constantly monitors for new JAUS connections. This allows the robot to navigate in autonomous mode seamlessly until JAUS connections are found, depending on the connection settings received from the JAUS enabled device. Once the JAUS enabled device is authenticated and requests control, our algorithm instantly switches from autonomous to JAUS controlled – once commands are issued via JAUS, the robotic platform can switch back to autonomous mode and begin executing the commands. This design ensures accurate connections with JAUS enabled devices, while still allowing the robot to perform in an autonomous manner – that is, our robot can now recognize, initialize, and configure new connections, receive commands, and return to autonomy without user intervention. This, we believe, is the ultimate goal.

4.9 Software Innovation

Archons software development contains three areas of innovations. These areas are: cross platform capability, generic algorithm, and JAUS integration.

From the concept of Archon, we knew that we wanted to create a cross platform program. One of the limitations of past robots is that there are different team members using different operating systems. This has proven to be a major challenge in the writing of the program; what may have worked on one user would fail on another. The concept of forcing a user to use a certain operating system was very unappealing, and the Qt language allows the user to pick and choose what platform they would like to use. The solution to this problem was to write a program that would work on all platforms. The language that we decided to use was Qt.

This language supports all major operating systems, including MAC, Windows, and Linux. Having a cross platform program also allows us some leniency in the hardware support for Archon. If we decide to change the hardware and operating system of Archon, then the Qt language will be able to be ported over seamlessly. This gives us great flexibility in our program design.

The second concept that we wanted to include in the development of Archon was to allow it to have a generic algorithm. This will also give us greater flexibility in future upgrades of hardware for Archon. A generic algorithm allows us to switch out hardware devices, such as switching an LMS with a stereoscopic vision system; this would not be a problem with our program. As long as the hardware device has a way to send data in a form that is readable by our program, it will be capable of seamlessly integrating into Archon. Qt, when paired with the generic algorithm, gives us a great deal of flexibility. We will be able to modify Archon without taking the hardware into consideration; whatever future technology is released, Archon will be ready to use it.

5.0 Conclusion

Archon is a first of its kind at Bluefield State College. Archon's original and unique physical design, as well as its use of mecanum wheels is things never attempted at Bluefield State College. With its strong path planning algorithm, we feel that Archon will be strong contender in this year's competition.