Oakland University Proudly Presents:

BOTZILLA

IGVC 2012

Student Members:

Micho Radovnikovich – Ph.D, Systems Engineering

Lincoln Lorenz – Ph.D, Electrical Engineering

Kevin Hallenbeck – Junior, Computer Engineering

Steve Grzebyk – M.S., Electrical Engineering

Michael Truitt – Senior, Electrical Engineering

Michael Norman – Senior, Electrical Engineering

Paul Abdo – M.S., Electrical Engineering

Oscar Vasquez – Sophomore, Electrical Engineering

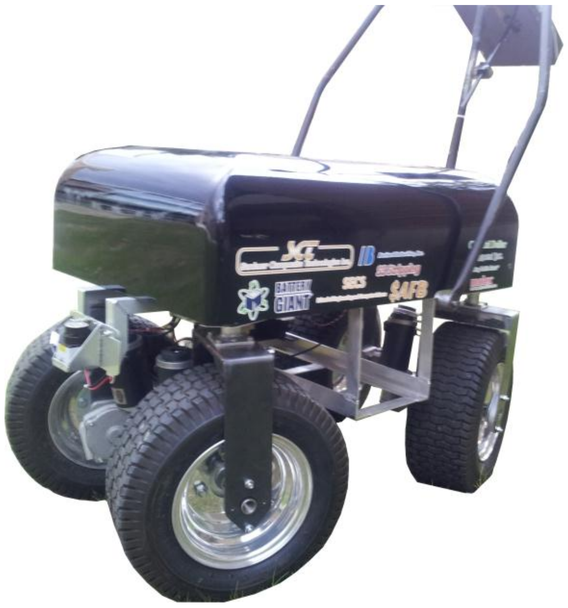
Kiran Iyengar – M.S., Systems Engineering

Ryan Bowman – Sophomore, Electrical Engineering

Raphael Parker – Sophomore, Computer Engineering

Joe Suriano – Freshman, Mechanical Engineering

Faculty Advisor: Dr. Ka C. Cheok



I certify that the engineering design present in this vehicle is significant and equivalent to work that would satisfy the requirements of a senior design or graduate project course.

Signed,

________________________________, **Dr. Ka C. Cheok,** Faculty Advisor

1 Introduction

Oakland University is proud to enter Botzilla into the 20th annual Intelligent Ground Vehicle Competition! Botzilla is a very rugged platform, featuring four-wheel drive and double Ackermann steering control **(Section 3.2)**. An FPGA is used to implement fast sensor data processing and high sample-rate control algorithms **(Section** 5**)**, and the computer vision and path planning algorithms are implemented in the very powerful and popular Robot Operating System (ROS) Environment **(Section 6)**. Significant improvements were made over last year's design, including a better-designed electrical system and much more sophisticated software and path planning algorithms.

1.1 Team Organization

The Oakland University team consists of 11 members, with a composition of about 50% graduate students and 50% undergraduate students, majoring in electrical and computer engineering. Figure 1 shows the organization of the team and how the responsibility is distributed. Each major sub-system of Botzilla has its own captain responsible for taking the lead role in its development. The captains were in charge of managing the rest of the group and guiding them towards completion of their respective components. It is estimated that about 1000 person-hours were invested in the development of Botzilla since the 2011 IGVC to modify and improve the mechanical, electrical and software systems.

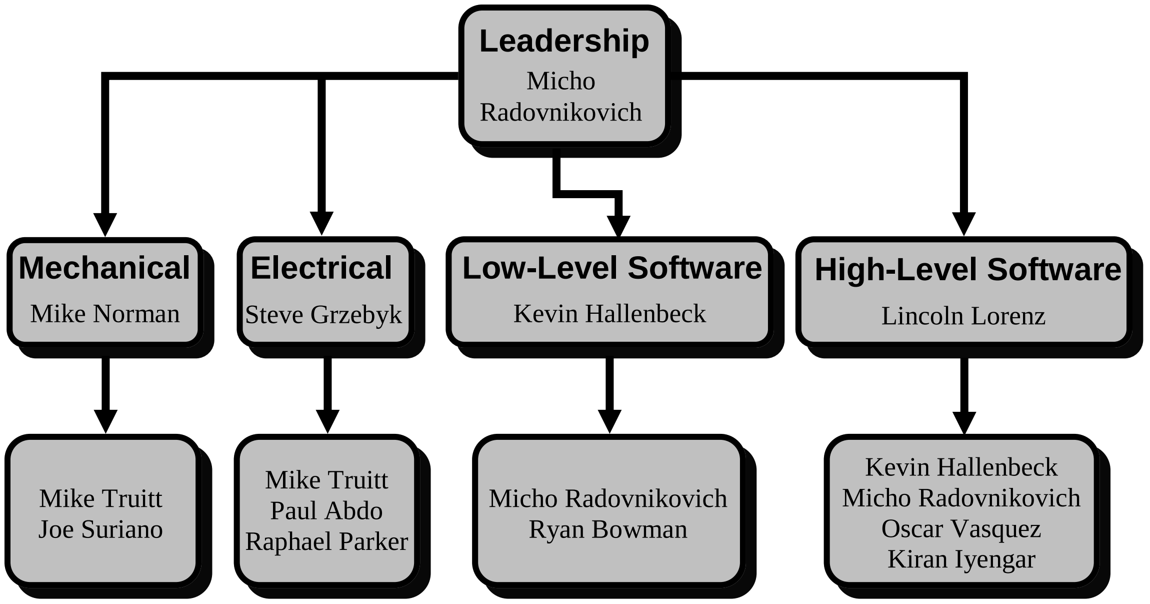


Figure 1: Chart outlining the team's organization.

1.2 Design Process

A classic ‘V-Model’ design process was followed to develop Botzilla, shown in Figure 2. After deciding on the concept of Botzilla, the requirements to achieve it were defined and a design was formed. After implementing the design and integrating the various components, a rigorous test cycle began, where consistent failure points were identified and rectified through minor adjustments or larger design modifications. To test the robot’s navigation system, a series of specific obstacle configurations were designed for the purpose of generating repeatable testing scenarios. This way, any changes in the software could be compared to previous versions by testing it in the same situations.

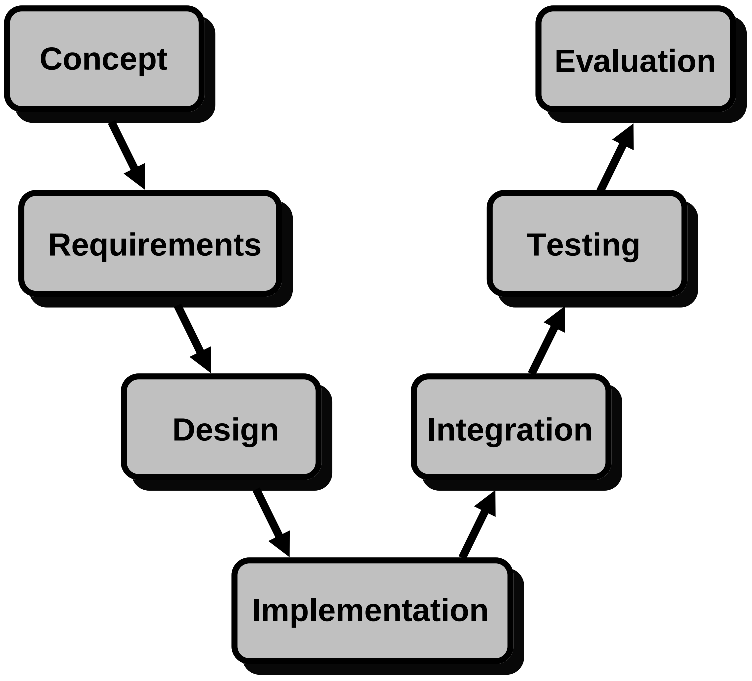


Figure 2: V-Model design process.

2 Innovations

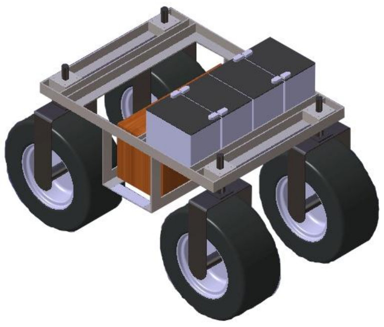
Below is a list of the main innovative aspects of Botzilla’s design. They are summarized here, and discussed in more detail in their respective sections.

- Monocular vision-based obstacle, line and flag detection **(Section** 7**)**
- Automatic camera calibration **(Section 7.4)**
- Robot Operating System (ROS) software integration **(Section 6)**
- Kalman filter-based sensor fusion algorithm **(Section 8.1)**
- SLAM-based mapping of the robot’s environment **(Section 8.2)**
- Custom H-Bridge design and fabrication **(Section 4.1)**
- FPGA-based sensor data gathering and drive control system **(Section** 5**)**

3 Mechanical Design

3.1 Chassis

The chassis of the robot is a simple ladder style design. This allowed for easy assembly and a rugged structure. The payload carrier is under the main chassis, allowing the quick installation and removal of the competition payload. A significant portion of Botzilla is built from aluminum in order to minimize weight. The use of steel was kept to a minimum; used only when strength or ease of fabrication dictated the need for it. All the electronics and batteries are covered by a custom fiberglass cover, which makes the body of the vehicle virtually weatherproof.

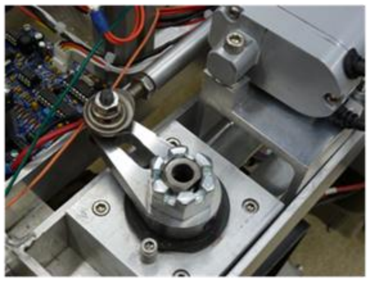

Botzilla's chassis

3.2 Drive Train

Each of Botzilla's four wheels has a brushed DC motor to drive it. The use of four-wheel drive allows for consistent locomotion while traversing irregular terrain. The four-wheel drive also allows the robot to ascend steep inclines with relative ease.

To steer, two linear actuators are used to rotate the front and rear wheels. Each linear actuator directly drives the left side, and is connected to the right side through a tie rod link. The link has opposing threads on the ends allowing alignment adjustments to be made. The double Ackermann steering allows for decent mobility in confined quarters while maintaining control of Botzilla under extreme conditions.

Chassis with cover


Steering mechanism

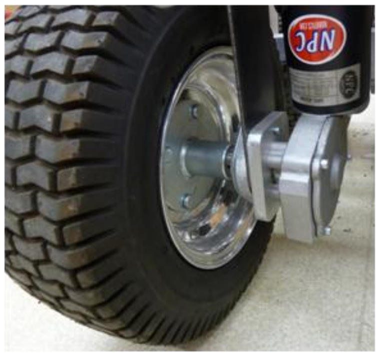

Drive Axle

Figure 3: Botzilla's mechanical design

4 Electronic Components

4.1 H-Bridges

Botzilla's H-bridges are completely custom-designed PCBs. Based on past experience with other H-bridges such as IFI's

Victor series, it was desired to use an H-bridge that is more flexible, robust, and capable of chopping the motor power at a much higher frequency. A conventional single-channel PWM signal controls the speed and direction of the H-bridge output.

Key features of the H-bridges are:

- Reverse battery protection
- On-board fuses
- Automatic fan control
- Over-current detection
- Temperature monitoring
- Serviceable components

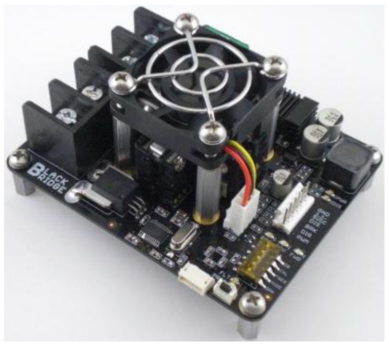


Figure 4: Custom H-bridge

4.2 Sensors

Botzilla is equipped with an array of sensors that allow it to detect obstacles around it, compute its location, heading and speed, and be operated in a safe and reliable manner. The sensor array consists of:

- NovaTel FlexG2-Star GPS receiver
  - 8 Hz, less than 1 meter accuracy
- MicroEye UI-155xLE USB 2.0 Camera
  - 1280x1024 resolution, 1/2" CMOS sensor, 8 bits per channel
- Hokuyo URG-04LX LIDAR sensor
  - 4 meter range, 240 degree field of view, 0.36 degree resolution
- Phoenix America P9122 magnetic wheel encoders
  - 800 pulses per wheel rotation
- DX5E wireless R/C aircraft joystick
  - Embedded controller-based manual control and wireless E-stop
- Honeywell HMC5843 tri-axis magnetometer
- Analog Devices ADXL345 tri-axis accelerometer
- InvenSense ITG-3200 tri-axis gyro

4.3 Computing Hardware

Botzilla's computing system is distributed between a Dell Latitude E5410 laptop running Ubuntu and ROS **(Section 6)**, and a Xilinx Spartan3E FPGA, employing a MicroBlaze soft processor and custom hardware modules written in VHDL **(Section** 5**)**. The two communicate with each other over a single RS232 link, through which the FPGA transmits sensor data to the computer, and the computer transmits motor control messages to the FPGA.

For the JAUS challenge, an external computer communicates with the Ubuntu laptop using and on-board IEEE 802.11g wireless router, and relays the processed JAUS commands. The Ubuntu laptop also responds back with the data reports necessary to satisfy the requirements of the JAUS challenge. The wireless router is also used to debug real-time software and adjust parameters on the fly.

4.4 Power Distribution

Botzilla's 24V power source is derived from four 12V AGM lead acid batteries, arranged with two parallel sets in series, with a total charge capacity of about 60 AH. The 24 volts are wired directly to the H-bridges through the E-stop, which then pulse power to the motors according to the PWM control signals coming from the FPGA. The FPGA board takes a 12 volt power input, coming from a switching DC-DC step down converter. The FPGA board has its own on-board switching regulator that powers the small sensors connected to it, as well as the Spartan3E itself. The 12 volt regulator also powers the GPS unit and the on-board wireless router. Another step-down regulator is used to provide clean 5V power to the Hokuyo LIDAR sensor.

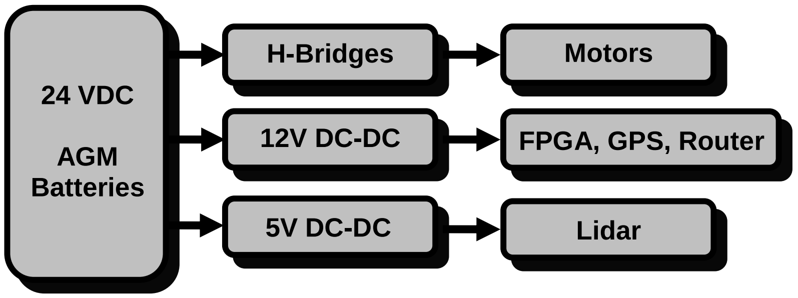


Figure 5: Block diagram of how power is distributed to the on-board components

4.5 Control Box

A waterproof control box, shown in Figure 6, is mounted on the camera shaft of Botzilla, and provides convenient operation of the vehicle in testing and performance scenarios. There are several switches that control power to various components inside the vehicle, including the GPS unit, wireless router, LIDAR and case fans. This makes the electrical system more power efficient by being able to turn off specific components when they are not necessary. A turn-to-release E-stop switch is also mounted in the box, and immediately cuts power to the motors when pressed. An Arduino processor is connected to a 4x20 character LCD screen, upon which it projects diagnostic data from the FPGA. Finally, the control box was designed to fit an

Android tablet, which is used as a remote control and quick parameter adjustment device. Figure 6 shows a picture of the control box.

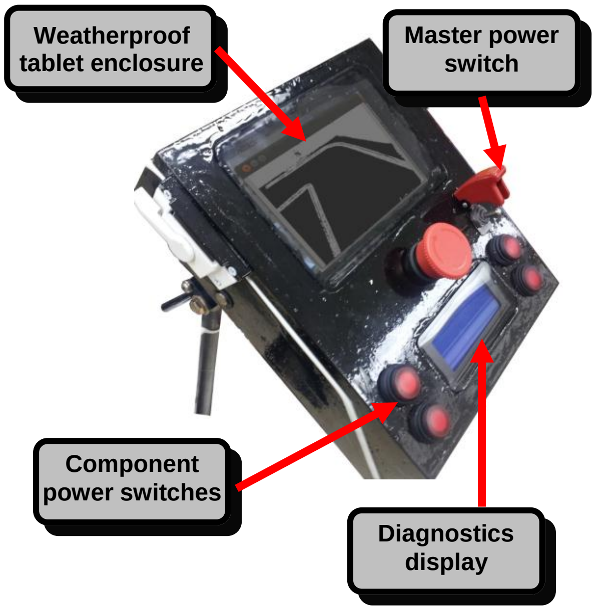


Figure 6: Botzilla's control box

4.6 Safety Considerations

Botzilla is a very powerful vehicle with lots of torque and is capable of slightly over 10 mph. As such, it has potential to be dangerous. Therefore, many precautions were taken into account when designing the emergency stop system, where many layers of protection exist. Besides the conventional turn-to-release E-stop switch, the DX5E joystick is capable of disabling the motor output wirelessly from up to a couple hundred feet away. In addition, the drive control system automatically turns off the motors if it fails to receive commands from the computer after a short time.

5 FPGA System

All of Botzilla's hardware interface functionality is implemented on a single 1.2 million gate-equivalent Spartan 3E FPGA, utilizing custom hardware components developed in VHDL, and a Xilinx MicroBlaze soft processor running C code. The system was designed to process the data from the sensors **(Section 4.2)**, extract the measurements, assemble the data into a convenient serial packet and transmit it all to the computer. At the same time, the drive control algorithms interpret vehicle motion commands from the computer and

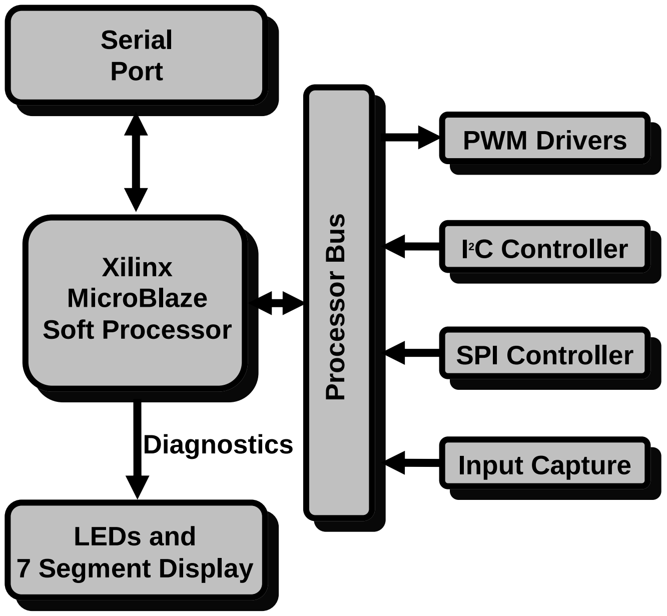


Figure 7: Custom FPGA Architecture

apply closed-loop control to the motors. Figure 7 shows a block diagram of the FPGA architecture.

5.1 Data Gathering

Custom hardware was developed in VHDL to gather the data from the sensors and drive the H-Bridges. These hardware components consist of:

- ❖ Six parallel PWM channels, capable of independent control of each motor
- ❖ One I^2C controller to grab data from the gyro, accelerometer and magnetometer
- ❖ One SPI controller used to interface to external A/D hardware module to measure the steering angles, and apply high-speed filtering to clean up the measurements
- ❖ Four parallel input capture channels to grab data from the magnetic wheel encoders

All the custom hardware is interfaced to the MicroBlaze processor by means of a 32-bit wide processor bus, clocked at 50 MHz. The processor addresses each hardware component in sequence and refreshes its measurement from each sensor. At a rate of 20 Hz, it packages all the data into a serial packet and transmits it to the computer.

5.2 Drive Control

Botzilla's drive system consists of four DC motors rotating the wheels of the vehicle, and two DC linear actuators controlling the steering mechanism on the front and rear wheels. Making use of the wheel encoder data being gathered by the FPGA, the MicroBlaze processor applies PI control to the wheel motors to match speed commands from the computer. Likewise, the potentiometers on the steering motors are used by the MicroBlaze processor to apply a lead-lag compensator to track angular position commands from the computer.

6 ROS Software Platform

Botzilla's software systems are implemented on the Robot Operating System (ROS) platform. ROS is an open-source development environment that runs in Ubuntu Linux. There is a multitude of built-in software packages that implement common robotic functionality. Firstly, there are many drivers for common sensors like LIDAR, cameras and GPS units. There are also general-purpose mapping and path planning software modules that allow for much faster implementation of sophisticated navigation algorithms.

6.1 Efficient Node Communication

An ROS system consists of a network of individual software modules called “nodes”. Each node is developed in either C++ or Python, and runs independently of other nodes. The nodes are all controlled by the ROS core. Inter-node communication is made seamless by a behind-the-scenes message transport layer. A node can simply “subscribe” to a message that another node is “publishing” through a very simple structure-based interface in C++. This allows for the development of easily modular and re-useable code, and shortens implementation time of new code.

6.2 Debugging Capabilities

One of the most powerful features of ROS is the debugging capability. Any message passing between two nodes can be recorded in a “bag” file. Bag files timestamp every message so that during playback, the message is recreated as if it were being produced in real time. This way, software can be written, tested and initially verified without having to set up and run the robot.

Another convenient debugging feature is the *reconfigure_gui*. This is an ROS node that allows users to change program parameters on the fly using graphical slider bars. This tool is invaluable, since most robotic vehicle controllers require precise adjustment of several parameters, and being able to change them while the program is running is very beneficial.

7 Computer Vision

Botzilla’s computer vision system is used in combination with a LIDAR to detect obstacles and lines in front of the robot. There are two separate algorithms used, one for general obstacle detection and another for flag detection. The input images of the system are first rectified using built in ROS functions. This compensates for the lens and camera sensor distortion making pixel coordinates linear with respect to the azimuth and zenith angular coordinates of the camera.

Since object detection in the robot’s environment is highly color based, both algorithms use the Hue Saturation Value (HSV) color space. The Hue component describes the color much like a human would identify colors of the rainbow spectrum. The saturation component describes the closeness of the ratio of the color information to the pixel’s intensity. Using HSV allows the systems to be relatively agnostic to lighting differences in an image. For instance, the color in the Hue component of HSV changes slightly from normally lit grass to grass with the shadow of an obstacle.

7.1 Line and Obstacle Detection

General line and obstacle detection for Botzilla is done using an adaptive median thresholding algorithm shown in Figure 8. Since the competition course consists of mainly grass, the algorithm essentially defines anything that is not grass in the image as an obstacle.

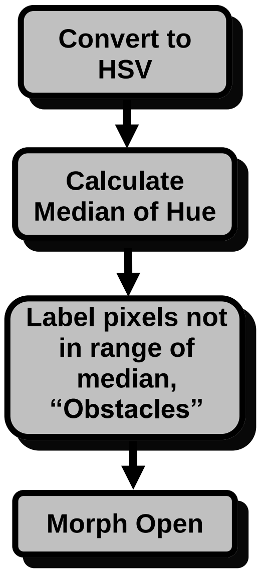


Figure 8: Median Thresholding

However, it was noticed that if the color of grass is defined to be a specific hue, some of the grass would be detected as an obstacle depending on the conditions. An adaptive median was added that calculates the median of the image and uses it as a replacement for the fixed value for thresholding. The assumption is that the main component of the image will be the grass. In some situations this is not the case. For instance, if the robot approaches several orange barrels the median will shift out of green and into the range of orange. To prevent this from affecting the detection, an enhancement was made that records the deviation of the median of a representative video of grass. This way the system can be trained with a realistic range of colors that grass can be. When the median moves out of this range, the system will replace its value with the last good value of the median.

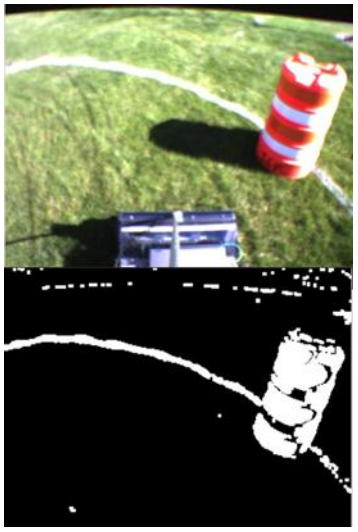


Figure 9: Sample Obstacle Detection

7.2 Flag Detection

Due to the different rules associated with flags, a separate flag detector was required for the system. Flags are distinctive using a combination of their size and color in an image. They are first detected by their color using a window thresholding algorithm, where the red and blue colors are each given a selective range that they are identified by. Pixels within each of the ranges are labeled '1' in their respective images. A max and min width of flags that appear in the image was determined experimentally.

To reject those red and blue components that fall outside these boundaries a morphological open is computed twice, once using a structuring element with the min width, once using a structuring element with the max width plus one. The results are exclusive OR-ed together to reject those that fall outside the range.

To remove noise introduced by the HSV conversion of pixels that don't have significant color, the saturation component is used. Pixels with values of saturation less than a certain constant value are not considered as flags. The results of a sample field of flags are shown in Figure 10.

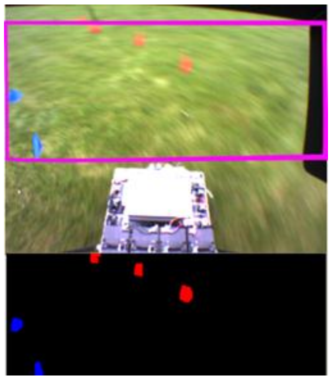


Figure 10: Sample Flag Detection

7.3 Obstacle Localization

After the obstacles, lines and flags are detected, the next step is to transform their position in the camera image into relative position from the vehicle. This is done through a calibrated reference frame transformation. The main assumption is that all detected obstacles are on flat ground, and that the bottom-most detected pixel in each column represents a point on the ground. An imitated LIDAR scan is then constructed by transforming each of these ground-level pixels into the vehicle reference frame. This is adjusted for the flags, since they are not positioned directly on the ground. In this case, an estimated height above the ground is assumed and used in the projection procedure.

Figure 11 shows a block diagram of the transformation procedure. The entire algorithm is calibrated by measuring 5 parameters: the height of the camera h, the azimuth and zenith field of view angles ψ and ρ, and the pitch and yaw orientation of the camera θ and φ. The values of h, ψ and ρ are relatively easy to measure, but the orientation angles are not so easy, and inaccurate measurements can throw off positional accuracy significantly. This complication was mitigated

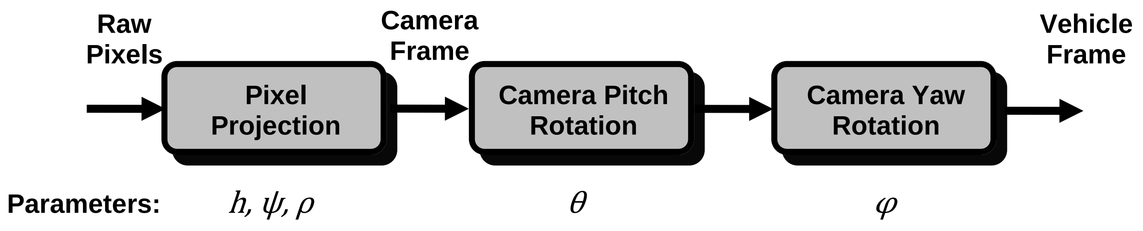


Figure 11: Object location transformation

by devising an automatic calibration algorithm.

7.4 Automatic Camera Calibration

Two objects are placed at known positions relative to the robot, and the pixel locations of each are set using the ROS *reconfigure_gui* node. A K-means based learning algorithm uses these known quantities, and recursively estimates the two unknown angles. The resulting calibration data has proven to be very accurate, with positions of detected obstacles being off by no more than 5% at any point.

8 Navigation System

A block diagram of Botzilla's navigation system is shown in Figure 12. The vehicle's position and velocity is estimated by a Kalman filter that combines the measurements from all the sensors. The position estimate and obstacle data are then fed into a SLAM mapping algorithm that constructs a map of the robot's surroundings, allowing the robot to remember what it has seen before. The global path planner uses the map and position estimate to construct a path to the current goal point. The local planner then decides on vehicle motion commands that move the robot along the global path.

Finally, a command bridge node is written to interpret the vehicle motion commands and translate them into Botzilla-specific control signals, while at the same time keeping track of waypoints, and sending the current waypoint to the global planner. Figure 14 shows a screenshot of the *Rviz* 3-D visualization program showing an example of how the path planners find ways around the obstacles.

8.1 Kalman Filtering

The Kalman filter performs sensor fusion, where individual, noisy sensors are combined to estimate the position and velocity of the vehicle. At the heart of the algorithm is a carefully derived kinematic state space model of Botzilla's double-Ackermann style configuration. The filter attributes how much noise is expected on each sensor, and how quickly the individual state variables change in the real world, and applies a set of recursive equations that estimate the state values based on current and prior sensor measurements. Figure 13 shows a block diagram of the Kalman filter algorithm.

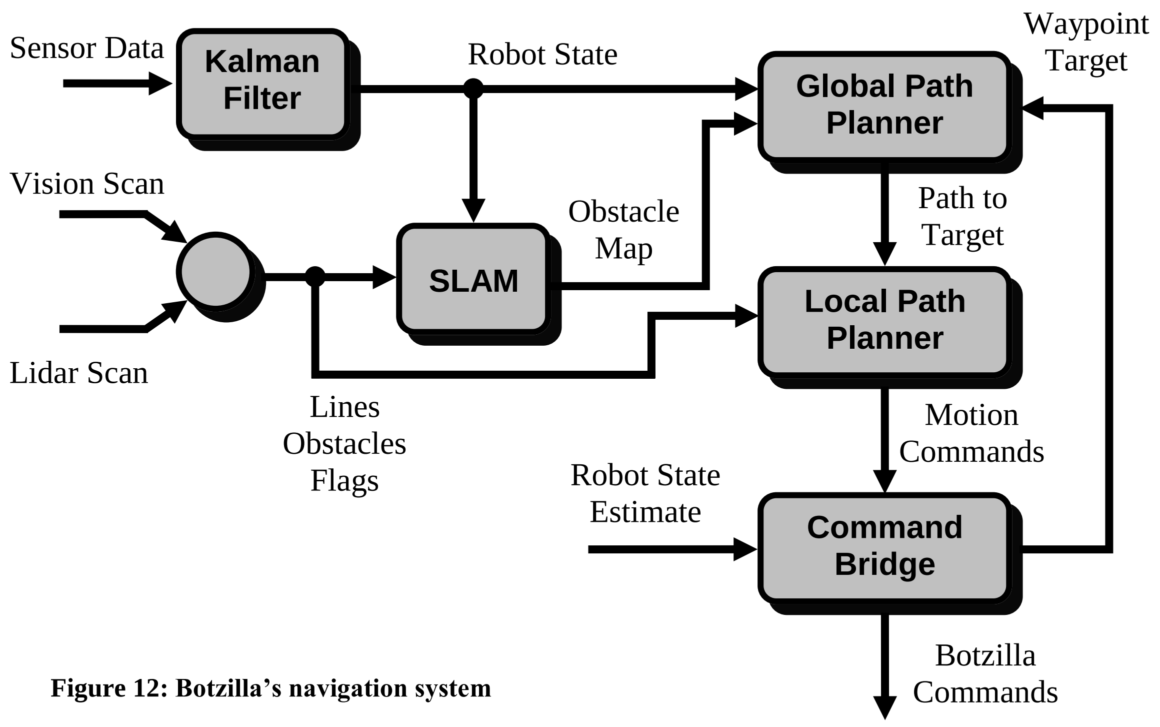


Figure 12: Botzilla's navigation system

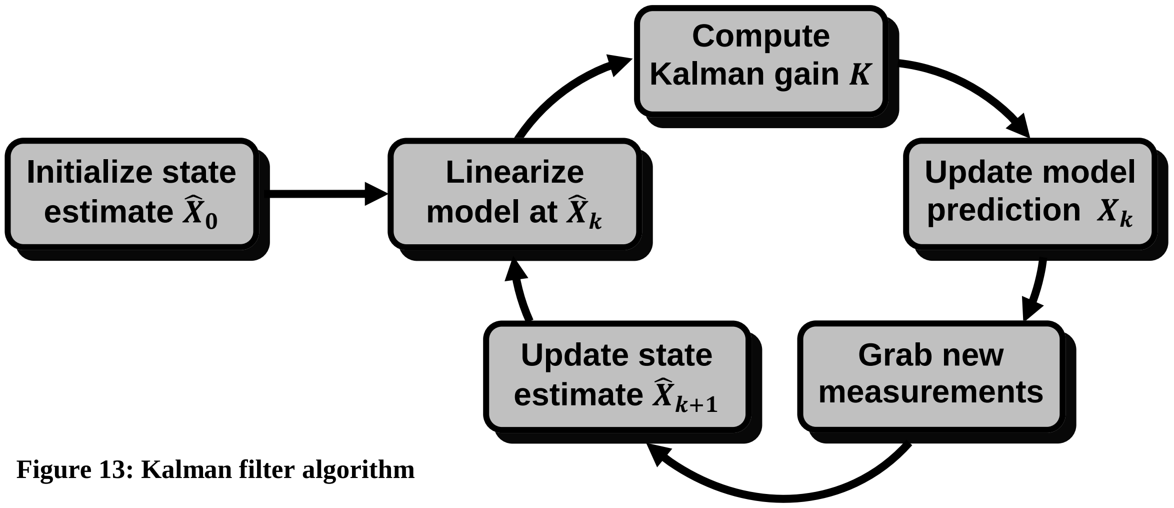

Compute Kalman gain K
Initialize state estimate $\hat{X}_0$
Linearize model at $\hat{X}_k$
Update model prediction X_k
Update state estimate $\hat{X}_{k+1}$
Grab new measurements

Figure 13: Kalman filter algorithm

8.2 SLAM Mapping

The Simultaneous Localization and Mapping (SLAM) algorithm is implemented in the built-in ROS package called *slam_gmapping*. This package accepts obstacle scan data from the object localization system **(Section 7.3)** and the Hokuyo LIDAR, and uses the position estimate from the Kalman filter. *slam_gmapping* then constructs a map by adding the new scan data, while correlating the scan data with the existing map to avoid double-mapping the same obstacles, and to re-align the map to account for gradual drift error.

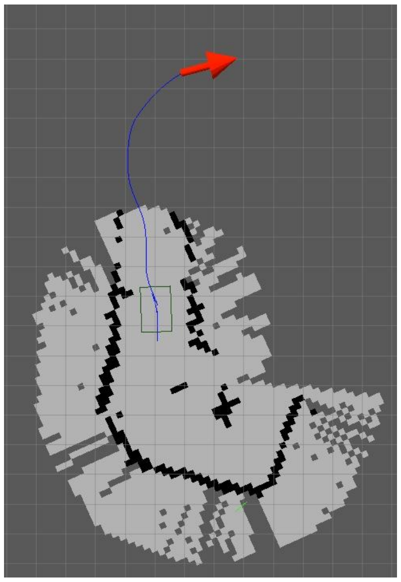


Figure 14: Visualization of the mapping and path planning systems

8.3 Global Path Planning

The global path planning algorithm is based on A*, and is part of the *sbpl_lattice_planner* package built into ROS. The planner takes in the state estimate from the Kalman filter, the map generated by *slam_gmapping*, and the location of the current waypoint from the command bridge, and solves an optimal path from the current robot position to the goal. This optimal path is generated from a sequence of feasible maneuvers based on the robot's dynamics. It then sends out this path as a sequence of points for the local planner to use.

8.4 Local Path Planning

The role of the local path planner is to follow the global path to the goal as closely as possible, while adding reactionary obstacle avoidance behavior. In addition, it detects scenarios where the robot is stuck, and finds a way to escape using the constraints of the map. The local planner is based on the *base_local_planner* ROS node, but modified to support Botzilla's specific requirements for IGVC.

8.5 Command Bridge

The command bridge converts the vehicle motion commands from the local path planner and turns them into individual motor speed and steering angle commands to be sent to the FPGA. The bridge also monitors the waypoint list and the vehicle's current GPS location, and communicates with the global path planner to generate paths to the current goal. It is written in C++ code and run as a ROS node.

9 Performance Analysis

9.1 Maximum Speed

Botzilla's motors spin at 157 RPM at nominal load, so combined with 15 inch diameter wheels, the resulting maximum speed is 10.3 mph. This estimate correlates with the observed performance.

9.2 Ramp Climbing Ability

At nominal load, the drive motors provide 101 in-lbs of torque. Assuming a realistic vehicle weight of 175 lbs, this corresponds to a max slope of 18 degrees. However, experiments have shown that Botzilla can handle much steeper slopes, up to about 30 degrees, although not at the nominal load of the motors.

9.3 Reaction Time

The ROS system running on the laptop gathers new sensor readings from the FPGA at 20Hz, and processes camera frames and extracts obstacle locations at 15 frames per second. The artificial intelligence systems were designed to be able to handle this frequency easily, thereby allowing the robot to make new decisions at the slowest sensor sampling rate 15 Hz = 66.7 ms.

9.4 Battery Life

The AGM batteries on Botzilla provide a total of 60 AH. The sensors and FPGA consume 2 amps. Experiments have shown that the steering motor current draw averages 3 amps under normal operating conditions, and that the drive motors consume a total of 25 amps maximum in a grass environment typically encountered at IGVC. Based on these observations, total battery life is approximately 2 hours.

9.5 Obstacle Detection Range

Using the monocular vision-based obstacle detection system **(Section 7.1)**, obstacles can be detected up to a maximum of 24 feet away, although it was experimentally determined that vision measurement data becomes most reliable within 17 feet. The camera configuration also makes the front of the vehicle visible. This allows for the vision system to detect lines and obstacles up to 3 feet to either side of the front wheels, thereby minimizing the size of critical blind spots. The Hokuyo LIDAR has a range of about 13 feet, but has shown to provide less noisy distance measurements at longer range than the camera system.

9.6 GPS Accuracy

Under normal conditions, the Novatel FlexPackG2-Star GPS receiver is accurate to within 1 meter, which is enough positional accuracy to reach the small waypoints on the GPS Challenge course. However, the Kalman filter algorithm **(Section 8.1)** fuses the GPS readings with the rest of the sensors to eliminate some of the noise.

10 Vehicle Equipment Cost

A breakdown of the cost of the components on Botzilla is shown in Table 1.

Table 1: Cost breakdown of components

Item	Cost	Cost to Team
FlexG2-Star GPS Unit	$1,000	$1,000
Hokuyo URG-04LX LIDAR	$2,375	$0[1]
Dell Latitude E5410 Laptop	$800	$800
MicroEye UI-155xLE Camera	$460	$460
Sparkfun 9-DOF Sensor Stick	$89	$89
Phoenix America P9122 Encoders	$152	$152
(4) 12V AGM Batteries	$375	$0[2]
Power Electronics	$600	$600
Fiberglass Body	$2,000	$0[3]
Electromechanical Components	$1,000	$1,000
Frame Materials	$1,000	$1,000
Total	**$9,851**	**$5,101**

11 Conclusion

Botzilla has proven to be very rugged, efficient and reliable, performing well while driving on any kind of terrain. The new artificial intelligence design shows promising results, and the Oakland University team has great confidence going into this year's competition.

Acknowledgements

The Oakland Robotics Association would like to thank several people and organizations. The team first of all thanks their advisor, Dr. Ka C. Cheok, for his invaluable advice and guidance. Thanks also to the School of Engineering and Computer Science and its dean, Dr. Louay Chamra, for generous funding and lab space to work in. Finally, thanks to our external sponsors Molex, Sankeur Composite Technologies and Battery Giant, whose donations of certain components of the vehicle were critical to its development.

[1] Re-used from previous vehicle
[2] Part of a larger donation from Battery Giant, Rochester Hills
[3] Custom-made and donated by Sankuer Composite Technologies