Michigan Technological University

Bishop Design Report

Ryan Bakker
Marcus Beltman
Jacob Henke
Mitchell Johnson
Tyler Novak
Joe Venier
Michael Wilson
Zach Wolbers

Contents

Introduction

The Blue Marble Security Enterprise (BMSE) project Autobot is sponsored by Oshkosh Truck Company for the Fall 2011 - Spring 2012 year. The purpose of this project is to improve upon a previously designed robot, "Ash", to compete in the Intelligent Ground Vehicle Competition (IGVC) in the summer of 2012. In addition to this, a second design "Bishop," is planned to develop throughout the semester. The competition has several different challenges. The team plans to participate in the Auto-Nav and JAUS challenges. These challenges require the robots to traverse an obstacle course involving obstacle avoidance, staying inside of lanes, and navigating to certain GPS waypoints. They also require implementation of the SAE JAUS standard for robot control and status reporting.

Team Organization

Over the past two semesters, eight people total have put an estimated 1500 hours into the project. Dr. Jeffrey Burl has served as the faculty advisor for both semesters.

Fall Semester:

Name	**Role**	**Major**	**Year**
Zach Wolbers	Project Manager	Software Eng.	Junior
Joe Venier	Documentation Chief	Mechanical Eng.	Junior
Marcus Beltman	Financial Manager	Electrical & Computer Eng.	Junior
Jacob Henke	Project Engineer	Mechanical Eng.	Junior
Mitchell Johnson	Project Engineer	Computer Eng. & Computer Science	Senior
Michael Wilson	Project Engineer	Electrical Eng.	Senior

Spring Semester:

Name	Role	Major	Year
Zach Wolbers	Project Manager	Software Eng.	Senior
Joe Venier	Documentation Chief	Mechanical Eng.	Junior
Ryan Bakker	Financial Manager	Computer Eng.	Sophomore
Marcus Beltman	Project Engineer	Electrical & Computer Eng.	Senior
Jacob Henke	Project Engineer	Mechanical Eng.	Senior
Mitchell Johnson	Project Engineer	Computer Eng. & Computer Science	Senior
Tyler Novak	Project Engineer	Computer Science	Sophomore

Estimated Cost

This year Autobot received a budget of $15,000 from Oshkosh Corporation. During this academic year, an entirely new robot was created. While the budget for both Fall and Spring semesters is considerable, the team stayed well within budget limits. The table below displays an overview of the budget.

Item	Estimated Cost
Batteries	$200
Cameras	$210
GPS	$30
IMU	$130
Laptop/Dock	$1,600
Laser Range Finder	$6,000 (Donated)

Linear Actuator	$130
Misc. Electrical	$150
Motor Controllers	$340
Motors	$700
Sheet Metal/Structural	$200
Wheels	$250
Total	**$9,940**

Hardware Overview

In the fall of 2011 we were asked by our sponsor, Oshkosh Trucking, to begin work on a new design to assure adequate engineering and design work was available for the project. While a software re-write was already planned, the idea of a fresh design was embraced by the team. The previous entry was showing its age, and nobody on the team had anything to do with the previous design.

The following goals were set for the new design:

- Serviceable design, easy to replace components
- Center of rotation aligned with the center of the robot
- Light weight
- Creative and different

A unique two wheeled balancing setup was settled on after several design ideas were scrapped for being too similar to our previous entry or other robots. All design goals were met in some capacity with this decision; however there were a few drawbacks to the design. It was not as strong as a the tested four wheel tub, and the addition of a third wheel would be a contingency if self leveling proved to be beyond our resources or abilities.

The two-wheeled design necessitated the use of an active balancing mechanism to keep the robot upright and stable. To this end, the design incorporated a linear actuator to swing the mass of the

batteries and payload relative to the robot's axis of rotation on a tray hanging from the wheel hubs. In this way, the robot's center of gravity could be shifted to keep the robot upright at all times.

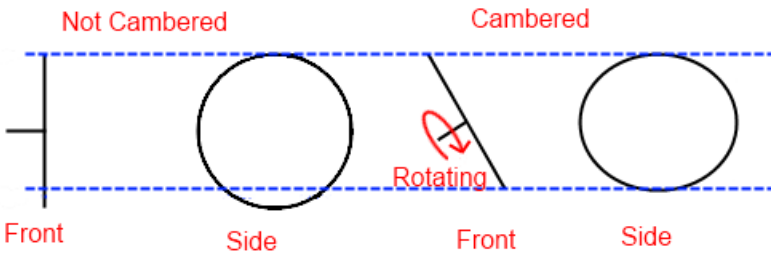
In practice, the actuator did not move fast enough to keep the robot balanced, so another solution was sought. Instead of moving the weight of the robot to compensate for changes in acceleration, the robot could be accelerated to compensate for a shift in its mass. The control theory was then similar to a Segway, with the added control variable of where the robot's mass should be placed.

To keep the robot upright, a simple PID filter was written. No special adaptations were used, as the system responded fairly well to a textbook PID implementation. The angular displacement of the robot was measured by an inertial measurement unit mounted on the robot's mast.

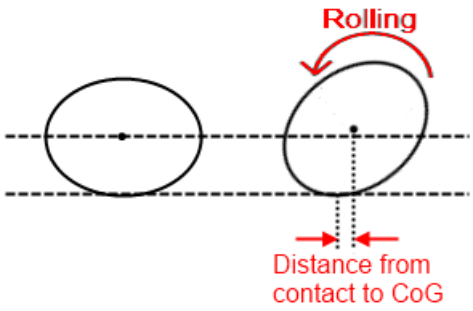
When the robot was taken outside, the small humps presented by the grassy surfaces presented an issue. The robot did not have enough rotated mass to account for the step in energy needed to roll over grass mounds. As such, when encountering a lump of grass, the robot would sit and oscillate, occasionally destructively. As a result, the team resorted to their backup plan and mounted a pneumatic caster on the robot. The caster provided ample stability without severely compromising maneuverability or maintainability.

Camber

One proposed design feature for the Bishop robot is mounting the wheels with a camber to further reduce rolling. The camber will make the profile of the wheels have an oval profile (see below), which will increase the robot's self-righting properties as increasing the amount rolling will move the center of gravity. The momentum of the robot will be counteracted by the height change by rotation, additional height from cambered rotation, and moment caused by the linear distance from the contact to the center of gravity. This extra force from camber will decrease the amount the robot will roll and shorten the time the robot will take to right itself.



The amount of energy necessary to roll the robot to a given angle is equal to the distance the center of gravity will be raised (which can easily be found through trigonometry) multiplied by the weight of the robot (also true without camber). See below for how rolling on cambered wheels moves the robot.



It is important to mention that in reality, the robot's center of gravity will not be at the center of the wheel as it is in the picture, since most of the robot's mass is in the components and tray. This means that the distance the center of gravity is raised by rolling, and the distance from the point the wheel contacts the ground to the center of gravity will be greater than shown in the figure.

Active Balancing

In order to adjust balance with this design the payload and batteries would be assembled on a secondary tray that would slide fore and aft with a linear actuator. This design was discarded in favor of a design that mounted the laser range finder to the main robot chassis. The tray mount design was also modified and instead of being fully affixed to the motor assembly the system was hung from the main wheel drive axles. In this way the batteries and payload could be fully fixed to the tray and the tray could be tilted from the motor assembly and act as ballast. By creating a longer pivot arm for the ballast, the linear forward and reverse distance was maximized which allowed for greater acceleration.

Custom Mounts

Much of the small mounting hardware for the robot was created from fully custom designs and printed in PLA plastic material with a small extrusion printer. The camera mounts are good examples of this. For

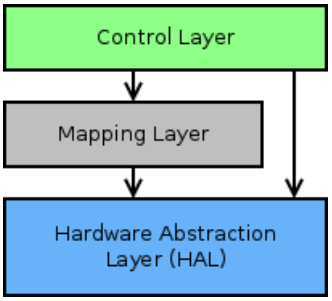
the visual image processing 2 cameras were mounted at the top of a mast. The cameras were delicate and not designed to be mounted except on top of a laptop screen. Plastic snap-together mounts were designed and printed that fit over the edges of the cameras to securely hold them in place. This is an innovation believed to be unique to this competition to date. Along with the camera mounts, a variety of T-slot mounting hardware, enclosures, and brackets were designed and created on the rapid prototyping device.

Software Overview

In the past, the team's software followed a rather simple algorithm. Sensor data was analyzed to determine which direction the robot could drive in the furthest; the robot would then point in that direction, drive forward, and reanalyze its environment. While this algorithm has done well in the past, it did have some major shortcomings:

- It disregarded some sensor data
- It did not perform complex path planning (for maneuvering around dead ends)
- Difficult to test; especially during the winter.
- Inherently sequential algorithm

In response to this, the team has worked to develop a new system to address these deficiencies. The software is broken into three main sections; the Hardware Abstraction Layer provides simple, low level access to the robot's equipment, the Mapping Layer processes current



sensor data to generate maps in real time, and the Control Layer sends movement commands.

Hardware

Simulation

When working with the team's past entry, one of the more surprising difficulties arose from testing. Effective outdoor testing requires favorable weather conditions (daytime lighting, no snow or hard rain, ect.), constructing an obstacle course, bringing the robot outside to the course, and finally testing. While none of these tasks are too complicated, they do introduce quite a bit of overhead for testing small, simple, 5 minute changes to the software. To help alleviate this, the team decided to include built in simulation capabilities in the software. Not only has this aided the team during normal development, but it has also allowed the team to perform comprehensive testing on the software while the robot itself was still being built.

In order for simulated testing to be effective, a few requirements must be met. Firstly, the simulated data must exercise as much of the software as possible. For example, the team can't simply plug in fake maps into the navigation software; this would leave all of the mapping software untested (approximately 40% of the software). On the other extreme, though, the team cannot practically simulate interactions with hardware (such as initializing the Laser Range Finder, or configuring the GPS). In order to achieve a balance between the two, the Hardware Abstraction Layer can be run in two different modes. When hardware is present, different components of the HAL will attempt to open, initialize, and maintain a device (such as the Cameras, or the Laser Range Finder). But when it's in simulation mode, these components will instead generate fake sensor data on the fly based of off a simulated environment. The same interfaces are provided to the rest of the software regardless of which mode the software is run in, allowing most of the same code to be executed regardless of whether or not a robot is attached to the computer.

Finally, in order to ensure the simulator thoroughly tests the rest of the software, random error is added to any generated data.

Line Detection

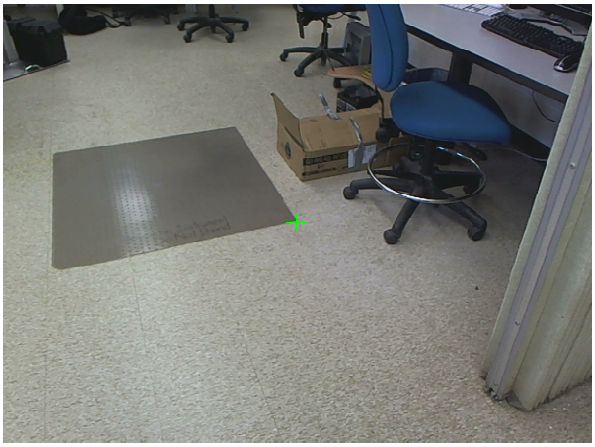
When pulling images from the cameras, the software needs to perform two main tasks:

- White lines must be detected

- Account for the camera's perspective (square objects on the ground appear slanted in the camera's images)

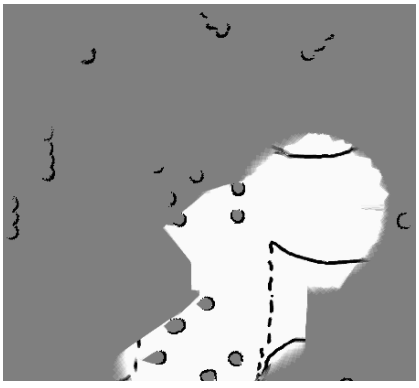
To do this, images are processed and manipulated using the library OpenCV. Lines are detected using a simple color filter, and images are warped to give the appearance of a bird's eye view (described below). Once these transformations have been applied, we're left with filtered, square images that indicate where lines are in relation to the robot's currently location, allowing the data to be easily processed by the mapping software.

In order to acquire data necessary for a perspective transformation, the team has created a simple calibration program. A 1x1 meter square is placed within the field of view of the camera and aligned to be square with the robot. Finally, the user simply needs to enter in the distance from the square to the robot, and click on the four corners of the square. The software will automatically determine the appropriate transformation, and output it's configuration into a file for use later.



Mapping

To generate maps of the robot's environment, the team is using an algorithm known as Iterative Closest Point (ICP) provided by the Mobile Robot Programming Toolkit (MRPT). Maps are stored internally as 2D point clouds, where each point represents a detected obstacle, and sensor data is also processed as 2D point clouds (such as those from the Laser Range Finder). As the software runs, it tries to iteratively guess how new sensor data



would best fit into the map. The hope is that while the majority of the sensor data would align properly, some of the data would not, adding new portions to the map.

In addition to the point cloud map, a second map, an occupancy grid map, is also generated for convenience. This map is simply a grayscale image, where each pixel represents the probability that an obstacle is present at that location (white = clear, black = occupied, grey = unknown). While this map isn't needed for the ICP algorithm to work, it does simplify the path planning process (see below), and provides visual aid for debugging purposes.

While maps can obviously have obstacles added to them, they can also have them removed. This can be useful as it helps filter out some sensor noise, but it does pose a problem with sensor fusion. Originally, the team was planning to combine both camera and Laser Range Finder Data together before it is sent off to the map building software. This would work great for adding items to the map, but since the camera's field of view is significantly smaller than the laser range finder's, camera data could potentially be removed from the map if it was out of view of the cameras, but not that laser range finder. To avoid this problem, the team decided to build two maps simultaneously, and merge them later.

While the ICP algorithm works well with the laser range finder data, the team quickly realized that it would not work for mapping the lines. Compared to the laser range finder, the cameras have a rather small field of view; it's entirely possible that while navigating the course, the robot will be unable to see any lines. Since ICP relies on aligning obstacle data, this would cause the map building software to fail, ruining the map. In response, the team developed its own, custom mapping software. Using the estimated pose of the robot in the obstacle map, camera data is scaled and placed on the appropriate position of a separate map. This should work well unless the obstacle map loses sight of obstacles for a stretch of time and the estimated pose is derived from odometry alone, but due to the laser range finder's large field of view, it's very unlikely that this will happen.

Navigation

Once maps have been built of the robot's environment, selecting a path is rather simple. First, the maps are merged by comparing the two maps (two grayscale images), and storing the darkest color in a third, merged map (transferring obstacles from each). Finally, the merged map is treated as a graph, allowing it to be processed using a well established path finding algorithm. While the team originally used Dijkstra's Algorithm for the task, the team eventually switched to A* for the increased performance.

Performance

Speed

Bishop's 29" bicycle wheels are powered by two 24V NPC-T64 motors with a 20:1 gear reduction. These motors were selected based upon durability and a mounting configuration which was suited to our design. Assuming a rolling resistance requiring 10 ft/lbs of torque to maintain maximum speed, the robot would be capable of traveling at 18.8 mph. This potential for high velocity is undesirable should something go wrong. Few devices on the robot required a 24v source, so the motors were run on 12v. We estimated this would cut the maximum speed, optimistically, in half. Considering a grassy field provides more rolling resistance than our ideal case, we could safely assume the robot would be hardware-limited to 10 mph. Encoders attached to the motors allow precise, configurable speed limits to be applied in software.

Ramp Climbing Ability

Travelling in reverse, the outboard caster keeps the robot from rearing up on high torque application. Under these circumstances, the robot is theoretically capable of climbing a 142% grade (55 degree slope), at 8 mph. This is assuming the tires have adequate traction on the incline surface. In practice, adequate friction could not be achieved to test this limit, however it seemed feasible. The position of the payload and battery can be adjusted on the fly to change the weight distribution of the robot such that an acceleration of 1.5 m/s^2 does not cause the front caster to lift off the ground.

Reaction Times

Bishop's software has profiling services built into it which prints off the times the software took to complete various activities. Based off these results, the team has realized that the overhead of mapping sensor data is insignificant compared to that of acquiring data. As such, obstacle detection is largely limited by our sensors, the cameras and the laser range finder, both of which operate concurrently at approximately 30hz. Unfortunately, though, path planning/navigation is more computationally complex. On average, paths can be computed in approximately .2 to .5 seconds, but we have observed worse case scenarios of about 1.5 seconds. In other words, while the software will detect obstacles quickly, it could take up to 1.5 seconds for the software to make an intelligent decision regarding this new data.

Battery Life

Component	Voltage	Current (to device)	Efficiency (est.)	Power
LIDAR	24V	1.7A	80%	43 W
Laptop + Peripherals	120VAC	.63A	65%	53W
Motors	12V	20A	95%	252W
Router	12V	1A	95%	13W
Misc.	12V	1A	95%	13W
			Total	374W

Efficiencies take into account losses in switching power supplies, and heating of wires and junctions, they are rough estimations. Using two 30 amp hour batteries, battery life is calculated to be in the neighborhood of two hours per charge. In practice, battery life lasts anywhere from 1.5 to 2 hours under operating conditions.

Obstacle Detection

For physical obstacles, the practical range of our laser rangefinder in typical conditions is 30 meters, accuracy is 1 cm, and the field of view is 180° at a .5° resolution. The cameras used to detect painted boundaries, have a combined field of view of approximately 135°, and a range of roughly 3 meters. White stripes on construction barrels show up on the map of painted boundaries, but these errors do not accumulate, they are removed from the map as soon as a new sensor acquisition causes conflicting observations, due to the barrels white lines existing in a plane which is not parallel to the ground.

Navigation Techniques

One of the benefits of using mapping/path planning is that “complex” obstacles (such as dead ends, switch backs, ect.) are treated identically to “simple” obstacles (such as straight corridors). Bishop performs with similar results in either situation.

Waypoint Arrival Accuracy

Bishop uses a single uBlox-5 GPS receiver, which can achieve a 2D accuracy of 2-3 meters. The projection of our GPS data onto our local map is run through a moving average filter to keep

atmospheric noise and jitter from causing undesirable perceived waypoint movement. In this configuration, observed drift is minimized, and destination arrival is based off of local map coordinates. In this configuration, the filter does not cause any major localization delays. In practice, destination arrival error has been observed at up to 2 meters from the desired destination. This needs improvement before the competition.

Simulation vs. Experimentation

Significant effort was put into developing simulation capabilities for Bishop's hardware. Major sources of noise and non-idealities are accounted for in simulation, including:

- LIDAR distance and angular error
- Wheel position error
- Camera position error
- GPS drift and noise
- IMU measurement offset and noise
- Sensor delay

Mapping algorithms were tested using different hardware simulation profiles, ranging from optimistic to worst case expectations for real world hardware performance, and desirable results were achieved. During physical testing, map building performed similarly to simulation with a couple of exceptions which are still unexplored.