Florida Institute of Technology & Florida State University

BEAST Design Report

Magna & Polaris Sponsored Project

May 15, 2017

Team Captain: William Nyffenegger (wnyffenegger2013@my.fit.edu)

Name	Email
Florida Institute of Technology	
Brent Allard	ballard2014@my.fit.edu
Adam Hill	ahill2013@my.fit.edu
Chris Kocsis	ckocsis2007@my.fit.edu
Rohit Kumar	kumarr2013@my.fit.edu
Matthew Salfer-Hobbs	msalferhobbs2013@my.fit.edu
Kartikeya Sharma	ksharma2013@my.fit.edu
Florida State University	
Ezekiel Copeland	ebc13@my.fsu.edu
Justin Daniel	jbd12b@my.fsu.edu
Andres Nodarse	aanb12b@my.fsu.edu
Matthew Patton	mspb12b@my.fsu.edu
Tajaey Young	tajaey1.young@famu.edu

Advisors: Dr. Matthew Jensen & Dr. Nikhil Gupt

Florida Institute of Technology
College of Engineering
Department of Mechanical and Aerospace Engineering

Florida Institute of Technology, Florida State University/Florida A&M University

Statement of Integrity

May 15th, 2017

Dear Intelligent Ground Vehicle Competition Judges,

I, Dr. Matthew Jensen, certify that the design, manufacture, testing and all associated work on the BEAST robot described in the accompanying design report to be of significant and equivalent work required to that of a senior design project.

Sincerely,

Matthew Jensen, Ph.D.

Assistant Professor, Mechanical Engineering
Program Chair, General Engineering
ProTrack Coordinator

Florida Institute of Technology
150 West University Blvd.
Melbourne, FL 32901
Office: (321) 674-7103
Fax: (321) 674-7270
mjensen@fit.edu

CONDUCT

INTRODUCTION

BEAST (Best Engineered Autonomous System Transport) is a vehicle designed, manufactured, and programmed by Florida Institute of Technology (FIT) and the joint Florida State University (FSU)/Florida A&M University (FAMU)'s Engineering program (hereby referred to as FSU/FAMU) as a collaborative project to participate in the 2017 Intelligent Ground Vehicle Competition. This is the first year the multi-university team will be participating in the competition, and lays the groundwork for future collaboration between the two schools. This paper documents the design process of the mechanical, electrical, and software systems, as well as safety measures and systems integration.

In conjunction with FSU/FAMU, Florida Institute of Technology IGVC (FIT IGVC) has developed a robot for this year's competition, as well as a long-term baseline for further development. FIT IGVC's contributions include GPU-based lane detection extendable to obstacle detection, a light-weight motion planning and mapping tool, a remote control system which seamlessly interfaces with autonomous control, and a flexible communication framework supporting software in multiple languages across multiple devices. FSU/FAMU focused on fabrication, motor control, position estimation, and obstacle detection. Both schools collaborated on mechanical design and power requirements. Hardware and software selection for the robot focused on novel hardware and techniques, including the NVIDIA Jetson TX1, the ZED stereoscopic camera, Sampling Based Model Predictive Optimization (SBMPO), and RabbitMQ. Other technologies used include: OpenCV, ROS, D* Lite, and the VectorNav 200 INS. Integration of the various components will continue until the competition in June. In summary, we have developed software components compatible with a unique hardware configuration, which is innovative, extendable, and capable of competing at IGVC.

DESIGN PROCESS

FIT and FSU/FAMU have spent the past two years researching and developing a robot for the competition. The first year was spent researching the problem space, defining tasks, and acquiring hardware.

This year the goal was to produce a competition worthy robot. The teams split tasks based on areas of expertise and resources. Florida Tech's team is composed of mainly computer scientists and electrical engineers, while FSU/FAMU's team is composed of mechanical and industrial engineers. FSU/FAMU also possesses more facilities and equipment for fabricating the robot. FSU/FAMU primarily worked on the fabrication, mobility, safety, and waterproofing of the robot. FIT focused on line detection, obstacle detection, course mapping, motion planning, position estimation, software engineering, software defined communication protocols, software interoperability ad system integration. FSU/FAMU and FIT collaborated on the mechanical design and power system design.

Communication between the two remote groups was a critical task. The team spoke via messaging services, phone calls, and occasionally travelled to meet in person. Slack, a commonly used professional business-messaging app, provides capabilities for business chats, private messaging, and file sharing. Channels focused on specific topics have been created as necessary. Several cloud services are also used to share important data. Microsoft OneDrive, GitHub, and GrabCAD were used to work together remotely.

INNOVATIVE DESIGN

INNOVATIVE TECHNOLOGY

Several core pieces of the robot are comprised of innovative technology. These technologies allow for major advances in performance, though sometimes at the cost of a very high learning curve. The first piece used was the NVIDIA Jetson TX1, a linux-based computer with a high-performance GPU. The TX1 is used to process our vision algorithms and motion planning. The second was the ZED stereoscopic camera. This camera can be used not only for standard images and video, but for determining the distance of objects in the camera's field-of-view. The third was the Quanergy M8 LiDAR. This LiDAR allows for 360 degrees of highly accurate distance measuring. The fourth and final piece was the VectorNav 200 Inertial Navigation System. This is a combination of an Inertial Measurement Unit (accelerometer, gyroscope, etc…) and a GPS, which allows for both position and orientation reports.

One of the most interesting things about the implementation is that given more time the LiDAR may be completely replaced by cameras. OpenCV provides the algorithmic capability to use a stereoscopic camera's data easily. Consequently, the price of this implementation may be significantly reduced in the future.

INNOVATIVE SOFTWARE

Perhaps the most innovative of our software techniques is our pathfinding and motion planning algorithms. This system combines two innovative algorithms, SBMPO and D* Lite, into a novel solution in robotic applications. SBMPO begins by selecting possible paths given the robot's model. It then calls upon D* Lite to select the best, most optimal path with that information. Once the path is selected it falls back to SBMPO, which takes the path as a list of motor commands and sends it to our motor controller. This eliminates the need for us to have a separate motion planning algorithm. The next software we use is RabbitMQ. This gives us the ability to send messages between different components of the software that need to communicate, even if they are written in different languages. RabbitMQ used a client/server setup that has a client for each language needed. The client can both encode and decode messages that are sent and received into a JSON format.

MECHANICAL DESIGN

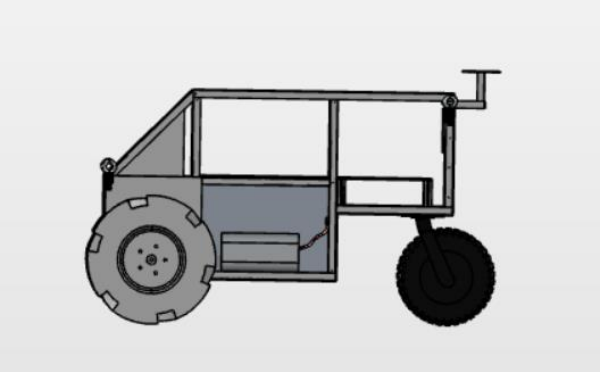

Figure 1: Early design

OVERVIEW

Three major sources of requirements influenced the design of the structural and steering components of the robot. The steering and durability of the robot were influenced by the competition course, which occurs outside in all but the most extreme weather conditions. Supplementing the course, competition documents themselves dictated motor and steering capabilities. Lastly, choices regarding computational hardware influenced structural components as well as the chassis. Priority was given to maneuverability

and survivability.

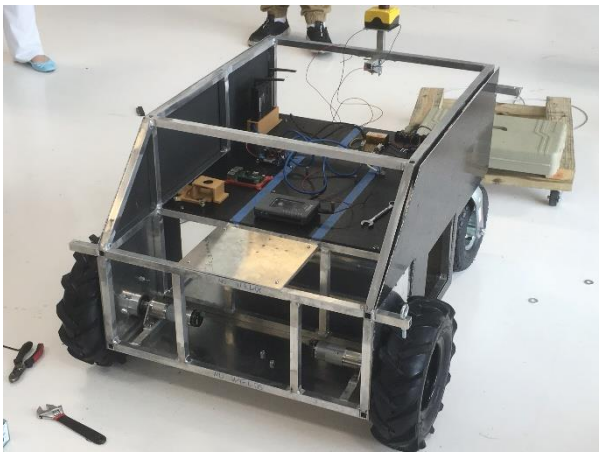

Figure 2: Fabricated robot

Concerning maneuverability, three steering options were discussed. These options included rack and pinion steering, differential steering, and skid based steering. Competition requirements specify that the robot must be able to maintain a tight turn radius and the turn radius itself influences software design. Rack and pinion steering was discounted because it limits said turning radius. Skid steering provides the ability to turn in place and use high traction treads; but, the complexity of the kinematics models is an obstacle the teams didn't feel ready to challenge. The complexity of the models stem from the different constants influencing steering based on the terrain (mud, grass, etc.). Differential steering is similar to skid steering except the vehicle does not typically lose traction making turns. From a programmatic perspective, differential steering performs in a predictable way, which the software can accommodate. Differential steering is implemented on the robot for its high turning rate, maneuverability considering the terrain, and simplicity. Two castor wheels were placed in the rear of the vehicle to aid in weight distribution and turning.

FRAME STRUCTURE AND WATERPROOFING

The vehicle itself was crafted using hollow aluminum square tubes for the structure. This allowed the vehicle to have a lightweight frame that could withstand physical abuse. To protect the electronics, carbon fiber panels were manufactured to provide protection from rain and other sources of water. The panels also allowed easy access to the systems. Most of the electronics were mounted onto a single platform that could be easily accessed by removing the top panel. Fans were added to increase airflow over the electronics to aid in cooling the various components.

For our motors, two PG27 Planetary Gearbox with RS775 Motors from AndyMark were selected. Based off of the given stall torque of 6.3 ft-lbf and the wheel placement we decided to use, it was calculated that 2 motors could move up to a 250 pound vehicle. Axles to connect the motors to the wheels were machined in the FSU/FAMU machine shop using steel, and bearings were press fitted into aluminum plates on the side of the vehicle to allow the axles to rotate freely.

To accommodate the 20 pound payload, clear acrylic panels were mounted inside of the front of the vehicle to provide a small channel for the payload to rest inside. A locking access hatch was installed at the front of the vehicle. This allowed the payload to easily be placed inside or removed from the vehicle.

Including wheels, the vehicle measures approximately 36 inches wide and 37 inches long. The overall weight without payload is 50 pounds. An aluminum mast is used to mount the ZED camera for line detection as high as possible, bringing the vehicle's height to 70 inches.

REMOTE CONTROL VS AUTONOMOUS MODE

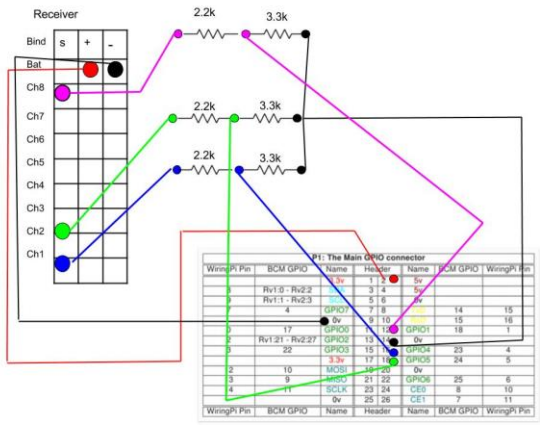

Figure 3: remote control circuit diagram

In order to remotely operate the vehicle, a Turnigy 9x remote control receiver is connected into a Raspberry Pi. The Raspberry Pi already handles communication between the myRIO and other subsystems, including the motion planner. Adding remote control to the Pi allows for seamless switching between modes. Additionally, adding a beacon light to fulfill a competition requirement to visibly indicate modes—RC and autonomous—is implemented by using the Pi.

The landing gear switch on the handheld controller is used as a Boolean control to switch between autonomous mode and remote control. From this, the Raspberry Pi may decide whether to pass autonomous commands to the myRIO or let the controller take over. This setup makes use of voltage dividers to prevent damage to the Raspberry PI. The WiringPi library provides functions to read PWM signals from the receiver and send commands via UART (Universal Asynchronous Receiver/Transmitter) to the myRIO.

SAFETY AND FAILURE POINTS

The competition rules required there be two different emergency stops for the vehicle. An emergency stop is defined by the competition officials as a way of bringing the vehicle to a complete and sudden stop. The primary stop is a wireless switch on the handheld controller. The secondary stop is an additional physical button placed on the robot in an accessible location, raised an acceptable height above ground level, typically on the rear of the vehicle. Both stops are required to be hardware and not software based. The design features a small platform for a large two-way switch to sit on. The platform and support are made of aluminum square tubes to withstand hard hits from people pressing the button. The button itself is wired into the RoboClaw Motor Controller. If the circuit is completed, then the power is directly cut from the motors.

The wireless emergency stop must work up to 100 feet from the vehicle. To accomplish this, two wireless Digi Bee devices are paired so if a button is pressed on one, the other signals the RoboClaw to stop.

The most likely point of failure on this vehicle is the motors for the wheels. Testing has shown that the motors are not as reliable as initially thought for all conditions in grass. This causes them to be stressed upon vehicle startup and overheat on occasion. The simplest solution for this is to minimize the operation time and maximize the cooldown time for the vehicle. Since this is not the most viable option, there are replacement motors available to install onto the vehicle if the need arises.

Other major failure points include the alignment of the wheels. This issue needs to be carefully checked in between runs. While having a wheel out of alignment won't cause the vehicle to lose control, it will make it harder for the motor control software to compensate and keep the vehicle on the most accurate track.

ELECTRONIC COMPONENTS AND DESIGN

OVERVIEW

The robot required several different hardware platforms to achieve the necessary results. The Jetson TX-1 was chosen to be the main computer platform due to its ability to handle large amounts of graphics and information. It worked in conjunction with a Mac Mini running ROS to merge the main components of vision, motion planning, and localization. Other important hardware includes:

- ZED Stereoscopic Camera: This is the primary vision sensor used on the robot. It is used for line and obstacle detection. The camera provides capabilities to sense the depth and color of its surroundings.
- VectorNav 200 INS/GPS: A 10-Axis MEMS IMU. This is used to track the movement of the robot and get its location. This GPS does not use a base station, which was a requirement of the competition.
- Raspberry Pi: This was used for relaying motor control information between the myRIO and the rest of the systems. The Raspberry Pi also handles remote control operations.
- NI myRIO: This was used as the motor controller of the vehicle. A PD controller was designed using LabView to be used on this device. Motor encoder values were read into the myRIO and PWM signals were sent out to the motor driver in response.
- Wi-Fi Router: This was used to create a local area network (LAN) among the devices on the robot for information exchange.
- Mac Mini: Ubuntu 14.04 device compatible with ROS
- SwiftNav Piksi: RTK GPS for testing

POWER DISTRIBUTION SYSTEM

The circuit is split into three parts and primarily powered through a 14.8 V 4S Li-Po battery. The two Jetson TX1s requires 12V to operate and exist on their own circuit with a 12V regulator. A set of smaller devices, including the Raspberry Pi and the router, exist on a 5V bus supported by a 5V regulator. The other devices are placed on a 12V bus and regulator; they consist of the heat management system (Fans), the Mac Mini and the MyRIO, all of which require 12V to operate.

Two main step down voltage regulators were chosen for this circuit, 12V and 5V regulators. We decided to go with step down regulators, as they are more efficient and stable. Our choice of a 14.8V battery made the dropout voltage became a major factor while designing the 12V bus. The dropout voltage determines how much voltage can be lost from the source until the regulator stops giving 12V constantly. The chosen 12V voltage regulator was selected as it had a low dropout voltage. This requirement ensures our battery will not go below 12.8V, as it needs to be recharged before hitting that voltage. The regulators also have capacitors built into them for noise reduction.

SOFTWARE / HARDWARE INTEGRATION

The two main computers used on the vehicle are:

- NVIDIA Jetson TX1
- Mac Mini

Line detection and obstacle detection are our main computational tasks. The primary concern regarding those tasks is the load created by the need for continuous image processing. The NVIDIA Jetson TX1 is built for image processing. It has a powerful GPU that can decode videos with up to 4K resolution (exceeding requirements). The board also has a USB 3.0 port, UART ports and built-in Ethernet and Wi-Fi for communication with other devices that we will be using. We also needed a stereoscopic camera for depth perception and object/line detection. The ZED camera from Stereolabs gave us all the specifications we needed. Also, the ZED has support on the Jetson TX1 through its Software Development Kit (SDK). We use two TX1s on the robot, one as the main hub and line detection, and the other one for obstacle detection.

Apart from the TX1, a Mac Mini provides compatibility with ROS (Robotic Operating System). We use ROS clients for the VectorNav INS and Piksi SwiftNav RTK for testing. The Mac Mini has Ubuntu 14.04 installed as its operating system for this purpose. Both computers are connected to each other via a local router, located on the vehicle, which routes all networking packets to their appropriate devices.

The final two devices, a Raspberry Pi and National Instruments myRIO, provide capabilities concerning motor control and remote control. The team has experience with these devices, which allowed a quick implementation; however, future teams may remove both in favor of the NVIDIA board.

Wiring Schematic

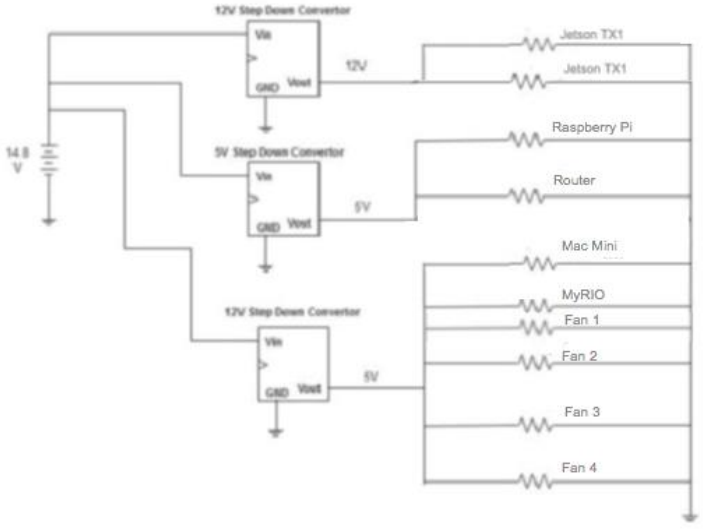

Figure 4: Circuit diagram

Safety and failure points

All the electrical components on the vehicle require a significantly low voltage and current to operate. Still, caution was taken in designing the circuit. Every component is powered through a voltage regulator, which keeps the voltage steady throughout the circuit. Also, all the components are connected in parallel so a failure of any one component won't result in an entire shutdown of the robot.

SOFTWARE PLATFORM AND STRATEGY

OVERVIEW

The overall software design methodology revolves around complexity introduced by:

- Multiple processes running time-sensitive operations
- Time-sensitive operations dependent on sensor data and abstracted sensor data
- Multiple programming languages
- Multiple hardware devices

Other constraints include the abbreviated timeframe of a senior design project, the research necessary to understand the software tasks, and the in-depth knowledge required of individual sub-problems.

The complexity and constraints led to two design process outcomes:

- An iterative design in which initial designs were refined by detailed research and experimentation
- The assignment of tasks solely to one individual on a team to allow development of subject matter knowledge

Through initial research, the team defined the likely software components, divided the components between schools and individuals, and made major architecture decisions. Experience led us to the most difficult components, where most of our efforts are spent developing.

The major challenges we defined are as follows:

- Developing motion planning software to traverse an unknown map and re-plan often
- Detecting lines and obstacles accurately and fast enough to influence the motion planner
- Building a map of the course in real time
- Determine the robot's position with enough fidelity to build a course map and reach waypoints
- Executing commands through a motor controller
- Communicating between processes in standardized formats across multiple languages
- Implementing an interoperable system, dictated by international standards
- Creating a GUI and logging tools for testing

Florida Tech has taken on the vast majority of these tasks.

SOFTWARE ARCHITECTURE

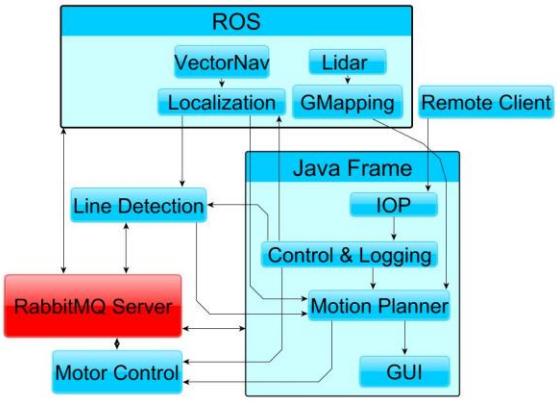

Figure 5: Software Architecture

A software architecture maximizing our experience and hardware has been developed to accomplish our challenges. The architecture focuses on the idea of independent components operating asynchronously without remote procedure calls, as dictated by a communication framework. Specifically, we wished to avoid becoming enmeshed in an approach with too much emphasis on a few massive components. Splitting the locations of components to maximize hardware was a major goal, as was maintaining a set of simple components.

The potential components defined include:

- A motor control client for interfacing with the motor controller software
- A motion planning and map building component producing motor commands
- A line detection component utilizing GPU programming; perhaps also accomplishing obstacle detection
- A localization component for producing high fidelity position estimates
- A GUI component displaying the state of the robot
- An interoperability component sufficient for the competition challenge
- An obstacle detection component if necessary using LiDAR
- An overall control component monitoring the states of every other component and changing those states as necessary

Over time we combined several components and introduced other components wrapped by ROS.

Languages and Libraries

Based on previous experience, the team chose Java to handle high level tasks, C / C++ 11 for low level control and sensory tasks, and CUDA to specifically handle line detection.

Java is used to accomplish the following tasks:

- Motion planning and map building
- Simulating courses and scenarios
- The GUI and logging services
- Abstract control of the robot

C/C++ are used to accomplish the following:

- Interfacing with motor control
- Obstacle detection
- Localization
- Line detection
- Sensor input
- Binding to CUDA

Libraries & tools heavily used include:

- WiringPi
- OpenCV
- RabbitMQ Client Libs
- GSON & RapidJSON

- JTS Virtual Lidar
- ZED Drivers
- Gradle
- Maven

The design of the architecture maintains the independence of most of the components to limit difficulties encountered with RPC, race conditions, or even the failure of components. Few components may give commands. The motion planning component issues actual commands for the robot to execute and control the GUI through which testing may be done. The interoperability component can relay messages from remote hosts to other components to change behavior including goals, modes, etc. Finally, the control component logs behaviors and can respond to situations regarding improper behaviors. The component can start, stop, and terminate components as well as change the frequency of publishes, etc.

MOTION PLANNING AND COURSE MAPPING

The motion planning module is responsible for tracking where obstacles are (mapping), calculating efficient paths to waypoints on the course (pathfinding), and generating a sequence of movements which are consistent with the constraints of the robot in question to such paths (trajectory generation), while processing a stream of detected obstacles. In order to ensure fast and reliable calculations, the D* Lite algorithm was paired with SBMPO to produce an adaptive pathfinding planner. It is required to dynamically calculate and recalculate the fastest way to get to the goal.

SBMPO, or Sample Based Model Predictive Optimization, is a technique that uses a model of a vehicle to produces a graph representing the space that vehicle can traverse. Because edges *(ways of getting between)* nodes *(locations)* are controls, which can be directly fed to the vehicle, applying SBMPO has a few advantages to alternative approaches. An SBMPO graph can be flexibly applied to a large domain of pathfinding algorithms, and is generated on an as-needed basis, preventing unnecessary waste of system resources. Additionally, SBMPO decreases the size of a pathfinding problem by reducing the search space considered to areas that can actually be accessed by the vehicle, according to its model. Most importantly, because the edges of an SBMPO graph are actions the vehicle takes to get from one place to another, pathfinding and trajectory generation occur in the same step. Finding a path will always result in an appropriately generated trajectory without extra processing.

D* Lite is an adaptive, heuristic-guided pathfinding algorithm which allows for efficient re-planning of paths to a goal, and is optimized for cases where the start point of the search may move, the target destination is fixed, and the graph is expected to change between recalculations. This makes it ideal for robotics applications. It works by discretizing free space into small grid squares, and keeping track of the optimal method to get to each grid square in that space. In exchange for the cost of maintaining this information, D* Lite allows a path to be recalculated when new obstacles are detected more quickly than one-off algorithms such as A*.

A traditional implementation of D* Lite treats grid squares which have an X and a Y dimension as individual nodes, and considers a grid square to be connected to those adjacent to it. This algorithm has been adapted for use with SBMPO by introducing a third dimension, the direction the vehicle is facing at the time of entering the grid square, and considering a particular node to be connected only to those squares it can reach according to the model of the vehicle. These two modifications provide the benefits of SBMPO, while also preventing certain circumstances which would result in a path not being found.

LINE AND OBSTACLE DETECTION

Line Detection Implementation

The primary goal of the line detection system is to precisely calculate the location of lines in as little time as possible. This goal led to the choice of parallelizing the processing of image data as much as possible. The tool used for parallelization is the Graphics Processing Unit (GPU) provided on the NVIDIA TX1 board which operates as the brain of the vehicle. The GPU provided by NVIDIA provides the speed necessary to process very detailed high resolution images. After experimentation, the team decided upon using an innovative stereoscopic camera called ZED to produce images which are then processed through the GPU. The ZED itself consists of two cameras, and a powerful suite of libraries assisting data analysis. It was designed with the goal of integrating with OpenCV directly on a GPU. The ZED's direct integration with the GPU allows the algorithms to be processed almost exclusively on the GPU where the calculations can be done in a much more optimized manner.

Implementing line detection begins by converting an image from the ZED camera into a the YCrCb image. A YCrCb image consists of three channels where the Y channel denotes the intensity or grayness of an image, the Cr channel denotes the redness proportion of the image, and the Cb channel denotes the blueness of the image. The Y channel is of particular interest because grass often reflects sunlight—much stronger than spray painted white lines, which may give false positives for lines. To remove those false positives, the Y channel is passed through a homomorphic filter, which removes high intensity contrasts from the image. The filter does so by transforming an image to a log domain (basically taking the natural log of the intensity values), performing another transform (FFT) to isolate peaks and then effectively chops them off before returning to the original image. The homomorphic filter removes high intensity areas in the image and normalizes light levels across the image.

Once the image is passed through the homomorphic filter, it is then converted to gray scale format to be passed through a Gaussian blur algorithm to assist in removing remaining noise that will primarily be coming from the grass that we will be operating on. The Gaussian blur algorithm takes square sections of the image and effectively blurs the colors slightly together to produce an overall smoother image.

The image is then passed through a Canny edge detection algorithm. This algorithm looks for contrast differences in the image and then determines and creates an edge on that contrast. This step is greatly assisted by the Gaussian blur, which removes many false edges that arise from ambient noise, such as blades of grass.

Finally, the image is then passed through a probabilistic Hough Lines detection. This algorithm produces a list of vectors that represent the found edges of the Canny Line detection. We can then use these lines, and also combine the two images from each camera, to get the best understanding of where the lines exist around us to then pass to the path subsystem to determine our course.

Obstacle Detection Implementation

Lessons from line detection led to a LiDAR based obstacle detection implementation. With advances in both cameras and LiDAR it is possible to take spatial information, pass that information through existing image processing libraries, and detect obstacles. For obstacle detection using LiDAR, both PCL and OpenCV were used. PCL was used to develop a general representation of the space that a 3D LiDAR provides and then transform that representation to 2D. The Hough Circles algorithm in OpenCV then extracts the centers and radii of circular obstacles on the course. A Gaussian Blur may be used as necessary to improve the performance of Hough Circles. The most difficult aspect of the implementation

is transforming a point cloud into a grid suitable for Hough Circles. Future implementations will work with just a ZED camera, Gaussian Blur, and Hough Circles.

COMMUNICATION FRAMEWORK

The communication framework was designed to work with multiple devices, multiple processes, and to be flexible concerning programming languages. These design choices reflect the complexity of an autonomous vehicle, the number of parallel operations, and the need for flexibility in the framework with evolving technologies.

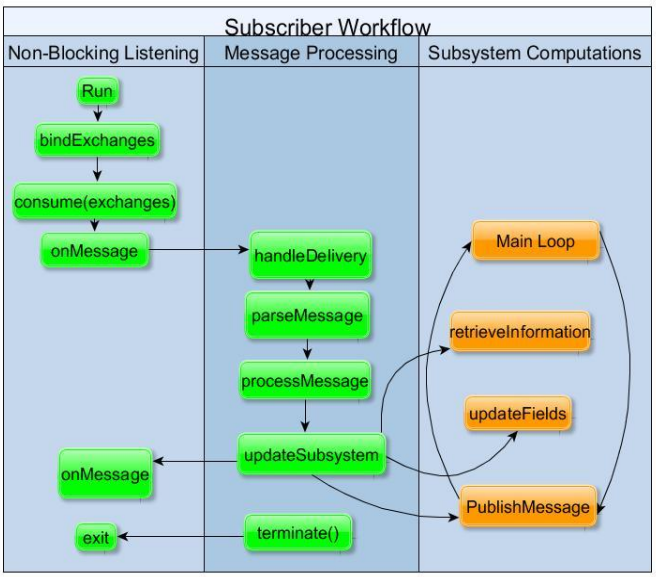

Figure 6: subsystem component behavior

An autonomous robot has many complex software components operating in parallel. Those components must process several different kinds of messages fast while maintaining service. Communication events should be processed asynchronously—in parallel with the running process. However, asynchronous processing loops introduce issues with synchronization. Furthermore, there is no guarantee that future projects will write code in the same languages or using the same tools chosen this year. Therefore, the framework itself must be generic enough to work with multiple languages.

With those considerations in mind, several tools were chosen and design decisions were made:

- RabbitMQ Server - a generic messaging server using the Advanced Message Queueing Protocol (AMQP)
- JavaScript Object Notation (JSON) as the standard messaging format
- Callback based message subscription and event handling
- Basic synchronization tools

RabbitMQ Server & JSON

The concept behind RabbitMQ Server is eliminating the complexity of communication between multiple devices, by having a central entity that handles routing, ordering, and delivering messages to different processes and devices. While RabbitMQ Server is on a device visible by all devices in a network, any process on any of those devices may communicate with any other process on any of those devices. Furthermore, RabbitMQ Server offers different routing protocols, allowing messages to be routed to one, some, or all processes using the server. However, routing is slightly different with RabbitMQ Server. Instead of directly sending messages between processes, messages are posted to the server with exchange-key combinations. Processes ask the server to forward messages posted with those exchange-key combinations.

For simple applications, programmers often define their own messaging formats. The complexity of our application required using a standard, well supported, flexible format. JSON was selected mainly for its flexibility in Java, where messages may be directly translated into classes using a concept called

reflection. Other benefits of JSON include its simplicity and the number of already existing serializing and deserializing tools available.

Implementation

The robot itself works around the idea of publishers and subscribers. A publisher is any program that publishes messages onto the server and a subscriber is any program that receives messages. Programs may be both publishers and subscribers; however, publishing and subscribing should be independent.

The framework was implemented in two languages with clients available via open source or officially from Pivotal, who authored RabbitMQ.

The Java implementation focused on producing a framework where important data was saved in synchronized structures available to a software component. Four components are in Java – the motion planner, the GUI, the events logger, and the interoperability client. These clients are all part of one process that splits components into threads and manages them via executor services and threads. The communication aspect of that means that each component has its own asynchronous, callback-based subscriber, which handles an event. The callbacks increase the resilience of the framework because exceptions occurring in the framework will not propagate to the connection.

The C/C++ implementation is not as clean as the Java implementation. RabbitMQ is generally used for web or phone applications. Normally those applications use languages like Python, Java, JavaScript, Ruby, etc. Support for RabbitMQ in C++ is more limited. However, several open source libraries do exist, though none are truly complete. Several such libraries were stitched together and wrapped to produce the necessary functionality. Another open source JSON handling library called RapidJSON is also wrapped.

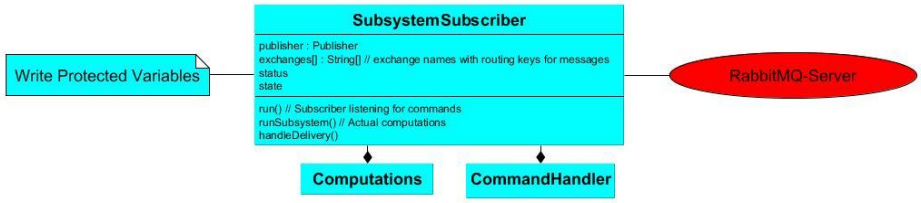

Figure 7: subsystem subscriber

On arrival of a message a callback is executed. That callback changes the state of data or flags available across the program which causes a change in behavior of the program. No data is explicitly returned in direct response to a message including interoperability messages.

SAFETY AND FAILURE POINTS

The complexity of the software system guarantees the presence of many failure points. Some of these failure points we obviated through our design, others are accommodated.

The software architecture assumes that a local network connection is maintained and that RabbitMQ-Server is always running. These assumptions are safe because we are using ethernet and RabbitMQ-Server will run unless the TX1 itself fails. What is more likely is that the client libraries that RabbitMQ provides for different languages will fail on connection issues in ways that are not easy to predict.

The most notable failure points concern the communication framework and C/C++ applications. Most sensors publish on specified intervals; however, some of those sensors are synthesized into position estimations so their behavior is critical. The most vulnerable components are the interface with motor control, and the communication framework's message subscribers. Poorly formed messages are harder to detect and handle in C++ than in Java. As a result, such messages could cause the failure of a component.

However, all messages are standardized with predefined classes, serialization methods, and deserialization methods. Someone somehow avoiding those predefined classes could cause significant problems.

Otherwise most of the software acts like independent publishers and subscribers, so failure of a single component will at worst cause the motion planner to give bad commands because the map it has built is missing information on obstacles and lines. Pausing the motion planner in that event obviates the problem.

SIMULATIONS

In developing code to test the motion planner, communication framework, and in general the Java framework, the team has devised its own simulations. Other teams commonly use Gazebo; however, we have developed software to create our own courses and test the robot under those courses. Initially, the team wanted to have a GUI running while the robot was running so its behavior was observable. The initial development of that GUI focused on the motion planner. That GUI has now evolved into an interactive simulation creator and executor. The simulation may be run physically on the robot and motor control executed. Components of the simulation may be replaced by actual components.

We are capable of simulating:

- Reporting lines, obstacles, location, and motor control execution
- Varying rates of reporting for those components
- Introducing latency
- Uncertainty in position, velocity, obstacles, and lines
- Different steering models
- Different search algorithms
- Different vehicle sizes, centers of mass, etc.

Most of these capabilities must currently be changed by hand; but, we anticipate adding options to the GUI.

ANALYSIS

SOFTWARE ARCHITECTURE

The team does not rely on pre-existing tools like ROS. This has allowed us to be flexible developing code. Not relying on ROS has been invaluable because without a pre-existing knowledge base concerning autonomous navigation we have been switching directions often. Furthermore, we have been able to modularize our software in a more flexible manner that fits the devices and technologies in use.

MOTION PLANNING

SBMPO as a motion planner has exceeded expectations. Using A* path planning may still be performed many times a second on large courses. The paths obtained are accurate while simplifying giving motor control commands. SBMPO handles a dynamic course seamlessly even when changes occur often in the course.

The only significant drawback to SBMPO is the requirement that a high-fidelity representation of the course be maintained. This requires a great focus on localizing the robot beyond what a GPS or even INS may provide. However, this is an inherent part of intelligently navigating a space.

OBSTACLE DETECTION

The performance of the line and obstacle detection software is impressive. Even evaluating images on a CPU, line detection may be performed at sub second speeds. Moving line detection onto a GPU is difficult because OpenCV's GPU implementations do not have up to date documentation and sometimes contain bugs. However, the speed improvements of moving computations to the GPU make the effort worthwhile. Obstacle detection is also relatively simple using the LiDAR. Perhaps the most difficult aspect of the LiDAR is transforming to a 2D plane for evaluation.

LiDAR's remain relatively expensive compared to cameras. The ZED camera provides very high performance and throughput to the point that it matches a LiDAR over short distances. While the camera may not handle dynamic lighting conditions well, practically, for this competition, a ZED camera could easily replace a LiDAR device and provide obstacle detection capabilities.

VEHICLE COST

The majority of the vehicle's cost results from the use of an INS and LiDAR. The frame of the vehicle was relatively cheap to manufacture, and the in-house facilities for carbon fiber on the FSU campus made it inexpensive to make the panels.

Part	**Price**	**Quantity**	**Cost**
ZED Stereoscopic Camera	450	1	450
NVIDIA Jetson TX-1	500	1	500
Apple Mac Mini	500	1	500
Raspberry Pi 2	40	1	40
Buffalo Wireless Router	60	1	60
PG27 Planetary Gearbox with RS775 Motor and Encoder	90	2	180
Large Castor Wheel	15	2	30
Wheels w/ Hubs	30	2	60
6 ft Aluminum Square Tubing	20	10	200
1/4 inch Aluminum Plate	50	1	50
Fan	10	4	40
myRIO	500	1	500
Carbon Fiber	200	1	200
Nuts and Bolts	20	1	20
Voltage Regulators	15	3	45
14.8v LiPo Batteries	50	2	100
VectorNav 200 INS	2500	1	2500
Quanergy M8 LiDAR	4995	1	4995
		Total:	**10470**

Replacing the LiDAR with a camera is entirely feasible and if done would significantly reduce the cost of the vehicle. The implementation of the motion planner also may allow replacement of the INS with a cheaper GPS and IMU combination. Those two steps would significantly reduce costs while still providing the same performance.

CONCLUSION

The platform designed for this year's competition is innovative in both hardware and software aspects. Though incomplete, the baseline set for the project, combined with the overall modular design, will allow improvement and expansion in years to come. The construction of the vehicle and development of software has all occurred in the past year.

The primary goal of the team—providing a complete map of the course to the motion planner—has allowed the team to introduce new navigation algorithms. The goal itself dictated use of advanced sensors and algorithms to provide a more complete picture of the course. The amount of information available enables the motion planner to build a representation of the entire course and navigate that course more intelligently. The techniques and ideas used parallel those in industry though they are of course more roughshod. With time this attitude toward the competition will become the norm. While the mechanical design of the vehicle is still in its infant stage, it provides an adequate vehicle for the purposes of this project.

The ambition of our design has made the robot more difficult to complete. Frankly, we may not finish the robot this year. However, the completeness of the design, especially concerning software, will provide a much greater end result with a greater potential for application. We have introduced new motion planning techniques, more advanced line and obstacle detection implementations, and a far more advanced software platform. Furthermore, we have strived to use the same technologies and techniques currently in use or being adopted by industry. Many of the sensors and boards we use are currently in use by industry.

Improvements to the performance of the vehicle will come with time and experience, especially considering that now there will be a pre-existing knowledge base. Priorities for next year include either developing or employing an existing Kalman filter, using cameras for line detection, and making the course representation probabilistic. Changes to the mechanical design will also be completed.