Florida Tech IGVC

Software Requirements

Intelligent Ground Vehicle Competition (FIT-IGVC)

Members:
Brent Allard ballard2014@my.fit.edu
Adam Hill ahill2013@my.fit.edu
Chris Kocsis ckocsis2007@my.fit.edu
William Nyffenegger wnyffenegger2013@my.fit.edu

Faculty Sponsor:
Dr. Marius Silaghi msilaghi@fit.edu

Contents

Introduction

The overall requirements for the robot consist of the basic functionality of any autonomous robot. These requirements extend to the software. The overall software architecture shall be modular; therefore, our requirements reflect that. The document is split into a series of sections each corresponding to one of the components of the software. For more extensive requirements search IGVC rules online.

Components reflect the major goals of the physical robot. These goals include:

- Construct a real-time map of a space
- Navigate that space efficiently and accurately
- Identify and potentially recognize "obstacles" and "roads" consistently
- Determine its precise location
- Perform many operations in parallel
- Execute motion across space
- Follow interoperability standards set by SAE

Based on these considerations and our need to test the software comprehensively, eight major subsystems are identified:

- A navigation and motion subsystem responsible for map building, path-finding, and motion planning
- A GUI displaying the software's conception of its location and path
- A vision subsystem responsible for analyzing sensor data to find and recognize obstacles, roads, and analyze the entire environment the robot is in
- A control unit responsible for starting subsystems, stopping subsystems and responding to subsystem failures.
- A communication framework responsible for maintaining communication between each of the aforementioned subsystems
- An interoperability subsystem to satisfy the JAUS requirements for the third challenge
- A simulation subsystem for testing the navigation subsystem, which may be extended to testing the vision and position subsystems

Rules Overview

The competition consists of three challenges:

- The navigation challenge
- The design challenge
- The interoperability profiles challenge

Teams are awarded points based on their performances for each of the challenges. The team with the highest overall score wins. The main challenges considered in this document are the navigation challenge and interoperability challenge.

The navigation challenge focuses on navigating the course. Points are awarded specifically for reaching waypoints, navigating past flags, the total distance navigated, and the total time used. Failing to complete the course does not mean that the run will not be scored. However, the robot must qualify on a qualification course before attempting the basic course. Points awarded for both interoperability and design may trump a high navigation performance.

The design challenge encompasses documentation of technologies and strategies for performing tasks. It is the natural paper outcome of following software engineering methodologies.

The interoperability profiles challenge focuses on the testability of the robot. Communication and behavior standards are defined in the JAUS document written by SAE. A subset of these standards must be implemented in the software. The standards define logging behavior for the robot, as well as the ability to send commands to and from the robot.

Basic Definitions

A series of definitions is presented which will be used throughout the document

- Component – one of the eight major subsystems underlying the software architecture
- RabbitMQ – advanced message framework that supports sending messages between different pieces of software
- ZED – the stereoscopic camera providing depth and color data
- Tegra – the integrated GPU board which all of the major software subsystems reside on
- "The robot" or "robot" refers to the entire software unit
- PCL – point cloud library, an open source library that understands and can reason over point cloud data
- SBMPC – Sampling-Based Model Predictive Control, the pathfinding algorithm being used on the robot

Vision

To traverse the course, the robot must be able to:

- Locate nearby obstacles in real-time
- Navigate to GPS waypoints indicated by GPS coordinates.

To accomplish the aforementioned tasks, the robot will utilize a ZED camera as well as computer vision algorithms to process the camera data in real-time. The image processing and vision algorithms will need to process large quantities of data in real-time and discern the shape of objects as well as the distance of said objects from the robot.

The ZED will produce several kinds of information:

- Stereoscopic raw image data

- Point cloud data, which is a 3D representation of the field of vision given in coordinates
- Depth information for the related points

The ZED camera will capture a single frame at a time, each frame will contain all of the data that the ZED can capture. The robot will be able to control the rate of information that the ZED will produce; the initial estimate is 15-30 frames per second will be necessary to gather enough information to locate objects surrounding the robot. This will produce hundreds of megabytes of data per second. To effectively process the large quantity of data the robot will utilize the Tegra GPU to process the data as quickly as possible.

The point cloud data that is produced contains significant amounts of noise that does not necessarily represent obstacles that must be avoided by the robot. The PCL library contains a variety of algorithms necessary for filtering the noise out of the PCL data and creating clusters of points and then for finally understanding and interpreting the bounds of these obstacles.

The overall goal for performance is to detect both lines and obstacles within several frames of data multiple times a second. That data shall be condensed down to a limited set of obstacle data, which the pathfinding and motion planning subsystem shall then use to build a map of the course. Because the vision subsystem is the basis for all other operations on the robot, it does not have many software dependencies. However, it shall require time sensitive use of the communication framework to maintain fidelity between the obstacles being listed relative to the robot's position.

It is also important to note that position and bearing estimation, a challenge that our team is not currently responsible for, might also become a challenge the vision module has to handle by using triangulation techniques.

Control

Starting and ending the robot requires initializing the communication framework, establishing connections with hardware, and starting individual software components in a regular manner. While the robot is operating the status of every component must be tracked. If a component fails, then all of the other subsystems must adjust. The failed component itself must be brought online.

The need for a central subsystem for monitoring and modifying the status of other components is amply necessary.

The control subsystem needs to accomplish the following specific tasks:

- Initialize all subsystems and prepare the robot to start
- Start all of the subsystems
- Monitor the status of each component and understand the overall status of the robot
- Change the behavior of each component as necessary
- Communicate with every component by receiving intermittent updates and sending standardized commands

- Respond to input from the GUI component and change the state of the robot
- Log the behavior of every subsystem during a run
- Respond to queries from the interoperability subsystem for the current behavior of the vehicle

Various status must be tracked and various actions must be taken based on the status of the robot's multiple subsystems. Specifically, the subsystem needs to receive statuses for each subsystem at least once every five seconds in addition to the data required for the interoperability challenge. These updates shall contain the status of the component—values depend on the component, if a component fails, the control subsystem shall be responsible for notifying all other components and restarting the failed component.

On command from the GUI or interoperability component, the robot will stop, start, and modify the parameters for components to the robot.

Concerning performance, the control subsystem shall conduct any of its operations in less than a millisecond. There will be no visible delay between receiving updates and responding to issues within any of the components. The performance of the control subsystem shall only be limited by the speed of the communication framework.

Messages from the GUI shall be the only commands that modify the state of the control subsystem. In the event that the GUI fails, the control subsystem shall reinitialize the GUI.

The control subsystem shall respond to inputs from the interoperability component; but, shall not rely on that subsystem to perform.

Communication Framework

The messaging framework must fulfill several primary functionalities dictated by the overall software architecture. Each independent piece of the software must communicate rapidly and often with other pieces of software. Requests for location, images, paths and motor commands must all simultaneously occur.

Definitions:

- A "command" is any request for information or behavior made by a software component of another software component
- A publisher sends commands to a server (RabbitMQ Server) which are then passed onto subscribers
- Queues are structures used to load and download messages from the server. Multiple messages can be in a queue at one time; however, using a single queue to send and receive messages shall not occur
- Exchanges are structures which the server posts messages on and which queues may publish messages to. Multiple other queues may listen to a single exchange and receive the same message posted on that exchange.

These constraints dictate a minimum set of capabilities for the communication framework which must:

- Support a publisher and subscriber structure implemented in each independent subsystem
- Be implemented in multiple languages
- Send messages to pieces of software in different languages
- Maintain a high throughput with many messages being passed
- State JSON contracts for the format and parsing of all messages in all used languages
- Define a basic set of commands implemented in each piece of software
- Reliably recover from the publication and delivery of poor messages

The messaging framework shall implement each of these functionalities by using the Advanced Message Queuing Protocol principles. Specifically, RabbitMQ Server shall be used to achieve high throughput and language independent communication. Furthermore, in each programming language used, a JSON command handler will be implemented which will define how to parse and encode every possible message that may be sent on the framework.

The framework will support publishers and subscribers in each independent piece of software. RabbitMQ has the capability to support exchanges and routing keys, allowing messages to be directed to specific recipients and filtered. The subscribers will be capable of running independent from the components they reside in to allow computations to occur despite receiving commands. Subscribers shall maintain threadable methods and callbacks to handle the arrival of messages.

Each independent component of the software shall implement an interface with the message framework based on already defined datatypes. That behavior shall include the ability to receive start, stop, restart, status, and terminate commands. Each independent piece of software must maintain a publisher to send and respond to commands as well as a subscriber to receive commands.

Commands shall be sent in JSON format as byte arrays and converted to utf-8 Strings before being parsed. Any message shall be receivable by any other unit of the message framework

Externally, the message framework only relies on being on a device that supports RabbitMQ Server. If the networking capabilities of the device are interrupted, then the framework could potentially fail (device's software would have to crash). The framework relies on being initialized once. After initialization, the loss of any component of software will not cause the framework to collapse.

Simulation

The simulation subsystem shall produce course simulations similar to the actual course. Courses shall be navigable by a simulated "robot", which the navigation subsystem shall attempt to navigate through the course.

This simulation subsystem shall:

- Create re-creatable, random course simulations complete with roads, obstacles, and goals
- Define obstacle interpretation and structure which is extendable to the navigation component
- Define structure and states of simulations
- Define metrics and methods for evaluating the performance of the navigation component on a simulated course
- Establish a communication standard between the navigation component for interpreting course information
- Provide time sensitive evaluation of the navigation components performance
- Provide time sensitive responses to requests to change the simulated robot's speed and direction
- Provide time sensitive responses containing what course simulated robot sensors see
- Provide updated data to the GUI on the performance of the navigation component and a visualization of the simulation

The simulation shall build and start simulations in under a second based on provided parameters detailing size and difficulty. The simulation shall use seeds provided by the user to generate repeatable simulations.

Running and terminating a simulation shall require one command. Providing frames—what the simulated robot can see, and maps shall occur in real-time. If data is too big to send over the communication framework, the data shall be stored in a file whose location shall be sent over the communication framework.

The datatypes and behaviors defined for interpreting obstacles shall focus on the concept of bounding boxes for obstacles. Obstacles shall be appropriately categorized.

Every time the navigation component either requests to modify or requests to update the simulated robot's position, behavior, speed, and/or direction, the simulation shall validate the navigation components proposed traversal of the course for violations.

If the simulation's rules are violated at any time, the simulation will send a status message alerting the navigation component and GUI.

The simulation component shall not rely on any other component aside from the communication framework.

Interoperability Profiles Standard

Standards for autonomous vehicle testing and interoperability have been established by SAE (The Society of Automotive Engineers) to govern the interaction of autonomous vehicles with simulations and devices on networks. These standards are named the Joint Architecture for Unmanned Systems (JAUS). For the purpose of the competition, these standards include communication standards for evaluating the performance of the robot, the status of the robot, and establishing basic security standards for whom may command the robot. The overall goal of JAUS is to ensure that all unmanned systems can be tested and evaluated for their performance using a common set of tools.

To satisfy these standards the robot must be able to connect to the Conformance Verification Tool (CVT) on a network. The CVT has been developed by the competition to test the adherence of the robot to JAUS. Connections to the tool must be made on a network using Ethernet or Wi-Fi to specific IP's and ports. Once connected communication must flow both ways between the CVT and the robot.

Communications from the CVT include commands to change behavior or report status; however, the robot is required to periodically report certain attributes. Judging will also involve another minor piece of software: The Common Operating Picture (COP), which shall record and display the robot's states as it accomplishes tasks.

To satisfy the JAUS standards, a subsystem to the control subsystem will be constructed that maintains a UDP publisher and listener. This subsystem will encompass the Platform Management and Navigation and Reporting Components mandated by the competition.

The performance requirements for the interoperability subsystem follow from the competition rules:

- Must qualify by completing handshake over locally hosted Wi-Fi, judge-hosted Wi-Fi, or physical Ethernet
- Responding to integer commands with messages
- Accept commands for changing the state of the robot (shutdown, start, etc.)
- Respond within one second of commands with relevant information
- Complete navigation tasks in the following areas:
    - Transport
    - Events
    - Access Control
    - Management
    - Liveness
    - Waypoint Driver
    - Waypoint List Driver
    - Velocity State Sensor
    - Local Pose Sensor / Primitive Driver
- Complete Platform Management Tasks:
    - Transport
    - Events
    - Access Control
    - Liveness
    - Discovery

A full list of tasks is included in the IGVC competition rules.

The interoperability subsystem will depend on an extensive logging standard built into the communication framework. Each subsystem will either need to periodically log its state or respond to requests to report its state. This will be discussed in the design document.

Pathfinding

Pathfinding is a core component of the system, which determines where the robot traverses. The pathfinding component will be the SBMPC / SBMPO algorithm, which integrates both global path planning and local motion planning.

A core challenge in pathfinding is going to be determining the location and bearing of the robot, a challenge which has been delegated to a separate part of the IGVC team. If we find that the information provided by their module cannot be relied upon due to sensor inaccuracy, modifications will need to be made to the algorithm to increase the likelihood of the robot avoiding obstacles and reaching the set goal.

The following functionality is necessary for pathfinding:

- Subscribe to messages sent from image processing about detected obstacles, boundary lines
- Updates map representation with low latency
    - Expected performance: constant time
- Uses a virtual model of the chassis, steering mechanism to sample possible motor commands
    - Algorithms predict where robot will end up after execution of said commands
- Finds a nearly optimal path to the currently set "goal"
- Sends sequence of commands to motor controller
- Commands are composed of (at minimum):
    - Linear speed
    - Turning radius OR angular velocity
    - Time to execute commands
- Reliably produces path to goal
    - Recalculates with different sampling techniques if one attempt fails

GUI

Fully understanding what is happening in each of the robot's systems during execution of simulations and course runs is a complex task. A graphical user interface will help the team observe the what the robot is "thinking", and interact with it in various ways.

At least two distinct GUI systems will be necessary to observe different subsets of the system at different times.

SBMPO GUI

- For testing performance and accuracy of SBMPO algorithm, parameter tuning, etc.
- Display results from running algorithm
    - Path found
    - Time, space usage
    - Branches created
    - The user-created map and the found path will be drawn
- Allows user to specify running parameters:
    - Branching factor
    - Number, size, shape, location of obstacles and goal
    - Size of map
    - Size of map grid subdivisions
    - Vehicle model
        - Chassis size
        - Turning Radius

Runtime GUI

- For usage with other software components, both on the robot and during simulations
- Displays a map of the course which will be updated in real-time with the following:
    - The robot's estimated position, bearing
    - Any observed obstacles, flags, boundary lines, waypoints
    - The robot's planned pathfinding trajectory
- IN SIMULATIONS displays a complete, fully accurate copy of the map for comparison
- Configurable panels displaying the following:
    - Vehicle metrics
        - Velocity, waypoints reached, etc.
    - GUI controls
        - Software-enforced commands such as KILL, PAUSE
    - Log of most recent information delivered to various components of the robot
- Doesn't drop any messages (display reflects all given information)
- Updates with minimal (<= .1 second) latency
- Persistently attempts to send messages representing user commands (button clicks) to other software components
    - The underlying technology for our messaging framework does not guarantee that a message the software attempts to send will actually be sent; however, it does guarantee that the failure to send or process a message is detectable

Faculty Advisor Signoff

Dr. Marius Silaghi: ____