FrostAV Development

Stage 1: Base

Updated November 5, 2019

Contents

1 System Abstraction

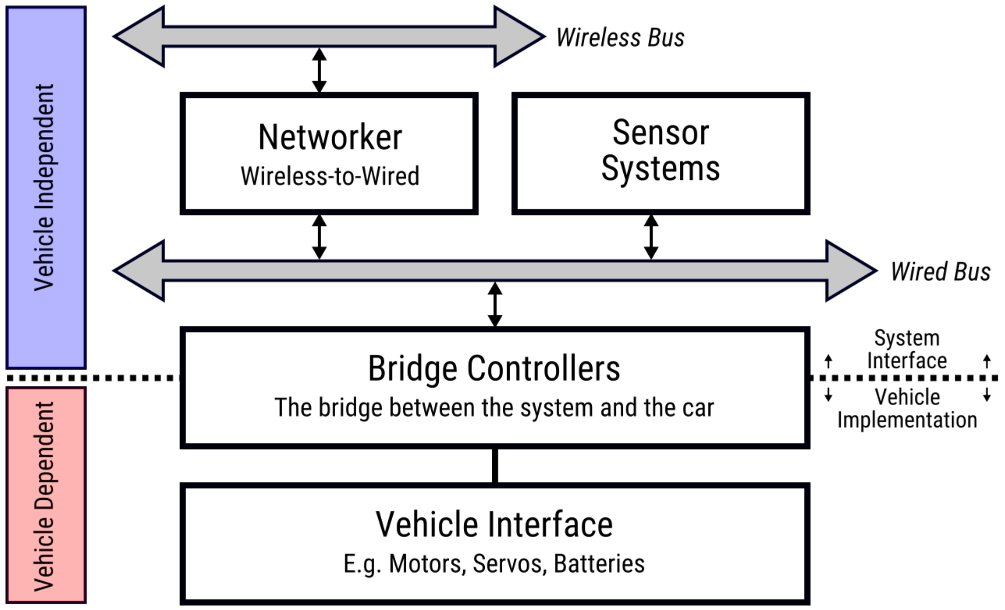


Figure 1: Fundamental module abstraction. Consists of a wired-bus for inter-communication between modules on the Frost vehicle, and a wireless-bus for communication with a server or an ssh client. Bridge Controllers act as the coupling between the vehicle and the modules that do not depend on the vehicle interface.

1.1 Abstract System Operation Description

The Frost Autonomous Vehicle consists of a vehicle and a system for controlling the vehicle. The vehicle provides an interface mainly to move it and to steer it. Bridge Controllers handle the peripherals that make up the vehicle interface. Bridge Controllers cannot operate unless they have instructions. These instructions come from a wired-bus. Different types of modules can send instructions to the wired-bus. The most important modules are the Networker and each individual Sensor System. Sensor Systems provide feedback for the Bridge Controllers to act upon. Furthermore, the Networker transfers information between the wired-bus and the wireless-bus. The wireless-bus allows for communication with devices that are not on the vehicle itself. Overall, the abstract system provides a map for how information is received and directed towards the vehicle's interaction with the environment.

1.2 Bridge Controllers

"Bridge" refers to a software design pattern from the Gang of Four, which intends to "decouple an abstraction from its implementation so that the two can vary independently." As such, the Bridge Controllers will have software that implements the Bridge design pattern, so that implementation code (for a specific vehicle) is decoupled from the system interface.

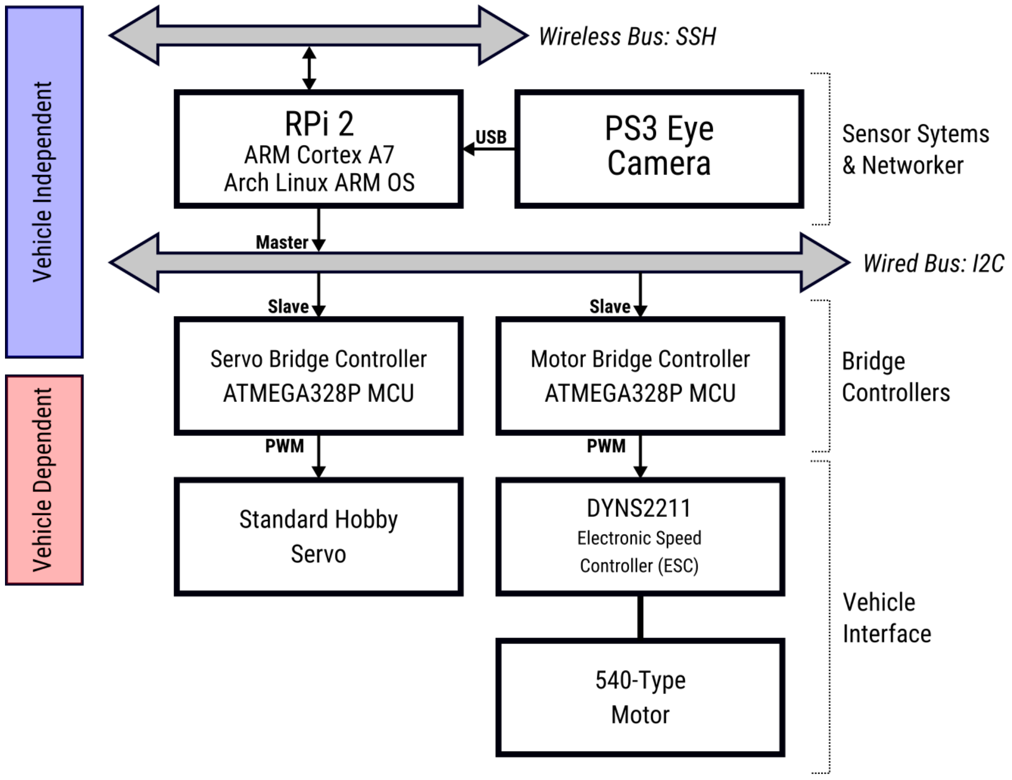
1.3 Networker

The Networker is reponsible for transfering information between the wireless-bus and the wired-bus.

1.4 Sensor Systems

Various Sensor Systems may be added to the wired-bus. An individual Sensor System should receive information from a sensor, process the information, and send the result to the wired-bus.

2 System Plan



2.1 System Operation Description

The PS3 Eye Camera provides visual information about the environment. This information will be processed with the OpenCV library on a Raspberry Pi 2 with an Arch ARM operating system. Processed camera information will then be piped through I^2C to the Bridge Controllers. The Servo Bridge Controller has a PWM output which directly drives the steering servo. The Motor Bridge Controller has a PWM output that is handled by the electronic speed controller (ESC), which drives the motor.

2.2 System Assumptions

The assumption made in this development stage is that the motor speed and servo steering angle can be controlled individually, such that motor speed and steering angle do not require direct knowledge of each other. Hence, the arrows from the wired-bus to the Servo and Motor Bridge Controllers are unidirectional. This assumption is meant to simplify the I^2C wired-bus configuration - since bi-directional communication on this bus would require arbitration.

2.3 Bus Communication

JSON will be used as the format for data for both the wireless and wired buses. JSON provides easy to understand data structure packets that can be serialized and sent as a stream. Using JSON sacrifices speed for readability and extendability of the system.

2.3.1 Wireless-Bus

In this stage the wireless-bus should allow for an SSH client to connect to it. As such, JSON packets that would normally be sent to the wireless-bus, will instead be stored locally in the system, but will be callable from an SSH client.

2.3.2 Wired-Bus

For the wired-bus, I^2C will be used. I^2C addressing will be done manually for each slave connected to the bus. Data will be sent from master to slaves via the JSON format.

2.4 Block Function Description

2.4.1 Raspberry Pi 2 (RPi2)

The RPi2 has an Arch Linux ARM operating system. It is responsible for processing camera information from the PS3 Eye. It is also responsible for being the Networker, which transfers information between the wired and wireless buses.

2.4.2 Servo Bridge Controller (ATMEGA328P)

Processes commands from the wired-bus to control the vehicle steering servo.

2.4.3 Motor Bridge Controller (ATMEGA328P)

Processes commands from the wired-bus to control the vehicle motor's speed.

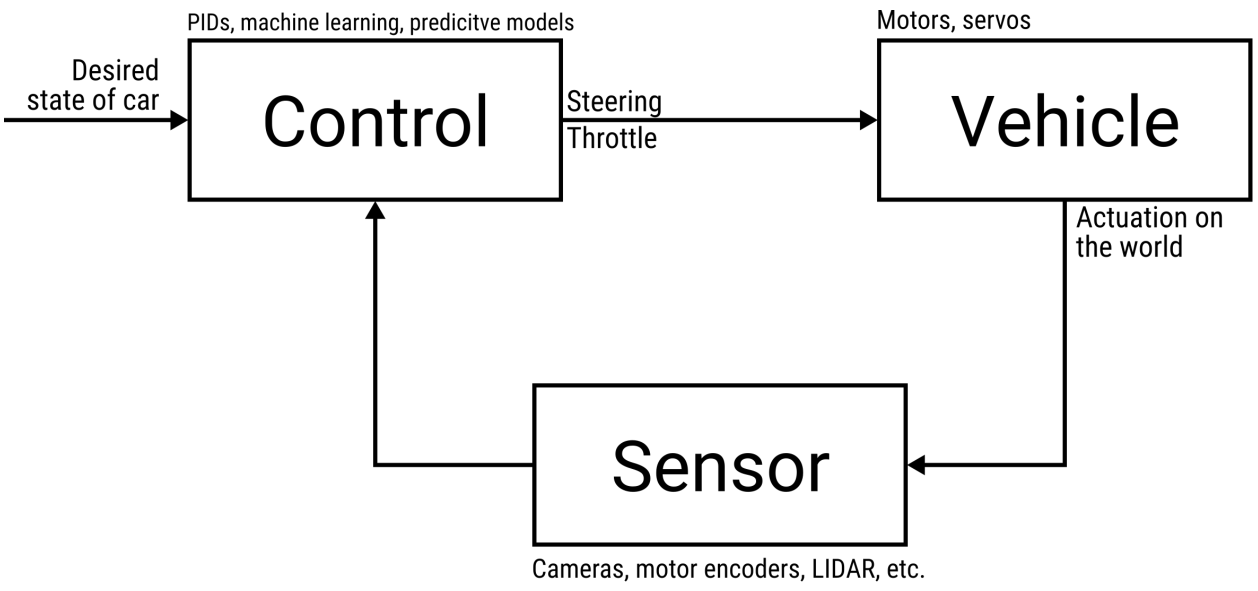
2.4.4 PS3 Eye Camera

Provides images over USB, for environmental perception.

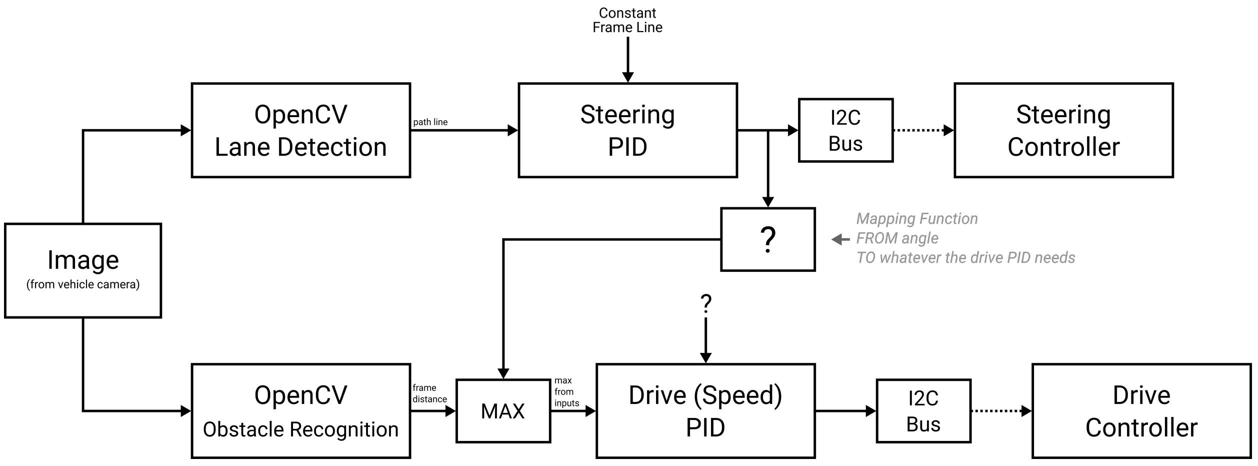
2.4.5 Electronic Speed Controller (DYNS2211)

Takes in a PWM signal and outputs an amplified and directional PWM wave to the vehicle's motor.

3 Control Abstraction



4 Control Plan



5 Future

5.1 I2C Arbitration

It is expected that modules communicating on the wired bus will need to communicate with each-other. In this case, there would be multiple masters on a single i2c bus. To implement this, arbitration would be necessary so that masters on the bus do not interrupt each other.

6 Steering PID

6.1 Concept

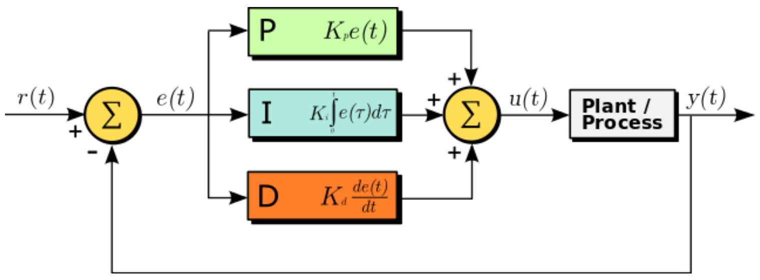


Figure 2: A block diagram of a PID controller in a feedback loop. r(t) is the desired process value or setpoint (SP), and y(t) is the measured process value (PV). *Arturo Urquizo, CC BY-SA 3.0 https://creativecommons.org/licenses/by-sa/3.0*

6.2 Parts

Part	Purpose
Arduino Uno (AVR attmega328p)	Steering Vehicle Interface Controller that interfaces between the Pi and the vehicle servo.
Standard Hobby Servo	The same type of servo that will be to steer the vehicle.

6.3 Experimental Setup

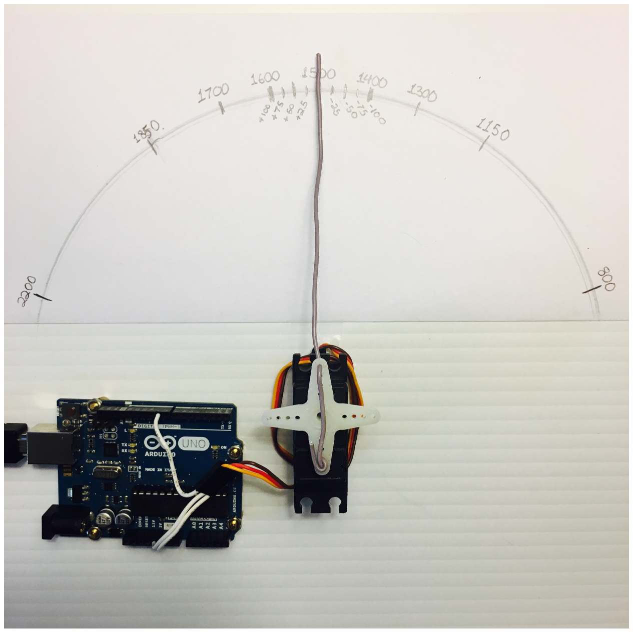


Figure 3: The Arduino provides microsecond PWM values from 800 to 2200μs to the servo. The servo reacts to an error value via a software PID controller. Since there is nothing in this setup that generates a true error value, error values must either be simulated or retrieved from a feedback sensor (such as a camera). The values along the circumference are microsecond values that correspond to the servo position; they help verify the accuracy of the PID.

6.4 Software Design

All code in this section was compiled using the avr-g++ tool, and using C++17. Test code was compiled with g++ as opposed to avr-g++.

6.4.1 USART to Command the Servo from a Terminal Emulator

In order to talk to the servo, we need to set up the USART on the AVR microcontroller.

6.4.2 Servo Control

To control the position of the servo, the attmega328p has a 16-bit timer. Once we set up the timer, we can provide it a pulse duration between the accepted values of the servo (typically 1-2ms).

6.4.3 Clamping

One common problem is providing a servo with a PWM value outside the valid range of the servo. For example, our servo has a maximum acceptable pulse width of 2200μs. If I provide our servo with a pulse width 2300μs, it could damage the servo. We will discuss a simple method for clamping the pulse widths provided to the servo.

6.4.4 PID

As discussed in section 6.1, a PID will provide reactive control to the servo from error feedback. In our system, the camera detects lanes. The further the car is from being centered between those lanes, the greater the error that is input into our steering PID. We've encapsulated the algorithm for a PID in a class.

6.4.5 Makefile to Facilitate Building, Running, and Testing

A makefile keeps builds up to date. It can also be as a general purpose script for shell commands, as we will see.