Velomobile Control & Telemetry System Software Requirements Specification

Version 2.0

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Ver.** | **Description** | **Author** |
| December 2, 2009 | 1.0 | Initial Composition | Daniel Johnson |
| February 2, 2010 | 1.1 | Template Modification | Shawn McGinnis |
| February 2, 2010 | 1.2 | Grammar, Phrasing, Data Dictionary | John Schmidt |
| February 3, 2010 | 1.3 | Graphics, Interfaces, Requirements Map | Shawn McGinnis |
| February 16, 2010 | 1.4 | Data Dictionary | John Schmidt |
| February 16, 2010 | 1.5 | Requirements Trace | Shawn McGinnis |
| March 3, 2010 | 1.6 | Updated for alpha release | Daniel Johnson |
| March 13, 2010 | 1.7 | Updated for alpha release QA | John Schmidt |
| April 18, 2010 | 1.8 | Logical & Component Architectures | John Schmidt |
| April 25, 2010 | 1.9 | Post Design Review Updates | John Schmidt |
| May 9, 2010 | 1.10 | Beta Update | John Schmidt |
| June 4, 2010 | 2.0 | Final | John Schmidt |

Table of Contents

[1. Introduction 4](#_Toc263447577)

[1.1 Purpose 4](#_Toc263447578)

[1.2 Scope 4](#_Toc263447579)

[1.3 Definitions, Acronyms and Abbreviations 4](#_Toc263447580)

[1.4 References 4](#_Toc263447581)

[1.5 Overview 5](#_Toc263447582)

[2. Overall Description 5](#_Toc263447583)

[2.1 Business Case 5](#_Toc263447584)

[2.2 Assumptions and Dependencies 5](#_Toc263447585)

[3. Requirements 6](#_Toc263447586)

[3.1 Functional Requirements 6](#_Toc263447587)

[3.2 Non-Functional Requirements 9](#_Toc263447588)

[3.3 Requirements Trace Matrix 10](#_Toc263447589)

[3.4 Purchased Components 10](#_Toc263447590)

[3.5 Interfaces 11](#_Toc263447591)

[3.5.1 Rider Interfaces 11](#_Toc263447592)

[3.5.2 Web Interfaces 12](#_Toc263447593)

[3.5.3 Hardware Interfaces 12](#_Toc263447594)

[3.5.4 Software Interfaces 13](#_Toc263447595)

[3.5.5 Communication Interfaces 13](#_Toc263447596)

[4. Use Case Model Survey 14](#_Toc263447597)

[4.1 Actors 15](#_Toc263447598)

[4.1.1 Interrupt 15](#_Toc263447599)

[4.1.2 Rider 16](#_Toc263447600)

[4.1.3 Web User 16](#_Toc263447601)

[4.1.4 Web Viewer 16](#_Toc263447602)

[4.1.5 AJAX Timer 16](#_Toc263447603)

[4.1.6 Telemetry Device 16](#_Toc263447604)

[4.2 Use Case Modeling 16](#_Toc263447605)

[5. System Architecture 18](#_Toc263447606)

[5.1 Functional Architecture 18](#_Toc263447607)

[5.2 Non-Functional Architecture 19](#_Toc263447608)

[5.3 Physical Architecture 19](#_Toc263447609)

[5.4 Component Architecture 22](#_Toc263447610)

[5.5 Logical Architecture 23](#_Toc263447611)

# Introduction

## Purpose

This is the introduction to the Velomobile system which covers the purpose, scope, definitions, acronyms, abbreviations, and references of the project.

## Scope

The scope of this project encompasses the embedded controller, referenced as ECU, design and implementation with the design and implementation of a telemetry web system. This document will be the “contract” used to determine our success at the end of the project. All requirements listed here will be completed in full by the end of spring quarter.

## Definitions, Acronyms and Abbreviations

This section will act as our data dictionary to aid users in reading the document. It will describe all terms, acronyms, and abbreviations used in this paper.

Analog: Transmission of a continuously variable signal, like the PWM signal.

ASP: Active Server Pages, used in writing and designing the web system.

Bluetooth Device: This project will use a BlueSMiRF for communication from the ECU.

C#: Microsoft Programming Language.

Digital: Transmission of a discrete level sensitive signal.

ECU: Embedded Control Unit, providing the base for the current regulation of the power assist.

GPS: Global Positioning System, relays the location of the velomobile via telemetry.

HTML: Hyper Text Markup Language.

Metric(s): The measurements or values from the sensors of the velomobile.

Power Assist: A high speed brushless motor, aiding the rider of the velomobile in its propulsion.

Power Assist Control: Mechanism between the battery and the motor, restricting current.

PWM: Pulse Width Modulation, the signal sent to the motor to determine speed.

RS232: Recommended Standard 232, a standard for serial binary data signals.

Telemetry: Data collected by the ECU and uploaded to the website for measurement and revision.

Telemetry Web System: A website that stores and displays telemetric data for viewing and revision via the website.

Velomobile: Recumbent Tricycle. Designed and built by the Mechanical Engineering team.

XML: Extensible Markup Language. The language accepted by the web parser.

## References

Competitors: Garmin. <http://www.garmin.com/garmin/cms/site/us/ontheroad/>

Web Server: XML SWF Graphs. <http://www.maani.us/xml_charts/>

Google Maps API. <http://code.google.com/apis/maps/>

ECU: Arduino. <http://www.arduino.cc/>

## Overview

The purpose of this document is to provide an in depth look at our system requirements, system architecture, and other documentation relevant to the project. By the end of the document the reader should have a clear understanding of the system we are building.

# Overall Description

This project involves the development of two systems allowing for the logging of telemetric data. The first system is the ECU for the Velomobile which will be the telemetric broadcasting device. Second being the Telemetry Web System which allows the logging and display of the recorded data. The ECU will perform the mathematical algorithms needed to control the electric engine power assist. Based on the power the user puts into the Velomobile through peddling, they will be able to pull an assist through the electric motor that will be proportional to the power they put in. The ECU will also display basic telemetry data and will stream its telemetry data to the telemetry web system we are developing. The telemetry web system will accept telemetry data dynamically from devices and store it in the database. It will then allow users to view real time and/or historical telemetry data.

## Business Case

The system is being produced because of the lack of an existing system which is capable of all current system requirements. The desire is to have all functionality incorporated into one simple to use, easy to understand system, while still having enough background to satisfy heavy/thorough users. Our goal is to implement a system with a touch screen, which will provide a simple display of system conditions and allow access to a web based telemetry system, possibly with web-feedback.

After searching the Internet, the only type of system which comes close to the requirements we are aiming to fill is Garmin. They offer GPS based search and navigation products for cars and motorcycles. However, these systems lack a web based data analysis aspect and offer no control scheme for power assist. Prices start at $125 and go up quickly from there to well over $250. Our system will incorporate some of Garmin's product's features as well as more specific features, such as control schemes for the power assist and the web end of data storage and review, designed specifically for our users.

From the functionality standpoint, most of the existing systems we would be using are too unlike our desired system to be feasible. Implementing small features on our website such as gauges and mapping utilities will be simple, as companies like Google allow for open use of their mapping tools. On the ECU, essentially everything not already a part of the basic OS will be done from scratch.

## Assumptions and Dependencies

This projects hardware launch will be dependent upon the functionality of the Mechanical based teams to design and build a functional Velomobile. On the web end of the telemetry system, it is assumed the user is capable of transmitting data to the web, this data must be in predefined XML formats deemed acceptable by our web presence.

# Requirements

The requirements section will cover the pieces of what the system will do. It will be broken into two parts, the functional and non-functional requirements. This should give a quick overview of what we are designing the system to be able to do.

## Functional Requirements

This section covers the “What” of our system. We'll go over what the system will be designed to do, without going into detail the implementation or “How” of the process.

1. **ECU Embedded Control Unit**

The ECU will control the power assist element of the Velomobile and provide data to the user of the Velomobile. It will also submit telemetry information, but for clarity we have left that out and cover it generically in the telemetry requirements.

* 1. **Sensor Input**
     1. *[Critical]* The System shall read and store sensor data for revolutions.

Revolutions are a critical piece of data that should be use in almost all of our calculations.

* + 1. *[Critical]* The System shall read and store sensor data for battery levels.

Battery levels will be used to provide raw information to the user and in other calculations.

* + 1. *[Critical]* The System shall read and store sensor data for voltage levels.

Voltage levels will be used to provide raw information and in other calculations.

* + 1. *[Critical]* The System shall read and store sensor data for throttle position.

Taking some form of level reading from the throttle, we'll record a position for use in assist calculations.

* + 1. *[Critical]* The System shall read and store sensor data for torque levels.

Taking readings from a torque sensor, the system shall store the information for use later.

* + 1. *[Critical]* The System shall read and store GPS latitude, longitude, and altitude.

*Take readings from a GPS sensor to store for later review.*

* 1. **Calculations**
     1. *[Critical]* The System shall calculate velocity using sensor data.

Using the revolutions and a clock internal to the system we shall calculate the velocity in meters per seconds and miles per hour.

* + 1. *[Critical]* The System shall calculate revolutions per minute using sensor data.

Using the revolutions and a clock internal to the system we shall calculate the revolutions per minute.

* + 1. *[Medium]* The System shall calculate battery life using sensor data.

With the information we have we should be able to give an estimate of the lifetime of the battery based on current usage.

* 1. **Power Assist Control**
     1. *[Critical]* The System shall apply power assist based on throttle.

Using the throttle assist we will be able to apply power assist to propel the Velomobile.

* + 1. *[Critical]* The system shall apply power assist based on sensor calculation data.

*Using the calculations retrieved from the velocity sensors, we will be able to apply power assist to help propel the Velomobile.*

* + 1. *[High]* Maintain velocity through assist when in the “Cruise-Control” mode.

When the user enters the “Cruise-Control” mode, the power assist should provide enough assistance to maintain the desired speed if the user is not applying enough power already.

* 1. **Embedded Display**
     1. *[Critical]* The System's default display will include primary data, velocity, battery level, etc.

The system's default display should display only vital information to the user, allowing them to get readings like the velocity, battery level, estimated battery life, or other such primary information.

* + 1. *[High]* The system shall have a method of displaying non-primary data to the user.

The system shall allow the user to view other data and information in regards to their Velomobile through an interface.

1. **Telemetry System**

The telemetry system is the second component to our system and will store telemetry data in a database which can then be viewed by users of the website that we will develop. These requirements outline the dynamic device sending the data, the web front end, and the back end.

* 1. **Device Data Stream**
     1. *[Critical]* Transmit a stream of telemetry data.

As long as the device adheres to our protocol standard, any device will be able to stream telemetry information of any kind to the web system.

* + 1. *[Critical]* Send all read sensors into the telemetry data steam.

This is applicable to the Velomobile, but taking all raw sensor data will hopefully allow easier testing and deployment of the Velomobile.

* + 1. *[Medium]* Maintain a list of clients that will receive the telemetry data stream.

A device should be allowed to transmit to multiple telemetry systems if needed.

* + 1. *[Medium]* Allow users to send “MARKS” into the telemetry stream.

Devices should be able to transmit time stamped marks in their data, which will allow users of the website to view segments of telemetry information, before, after, or between marks.

* + 1. *[Medium]* Send encoded video frames into the telemetry data stream.

Devices should be allowed to transmit video frames to the server, so as to have a live video stream displayed from the device.

* + 1. *[Low]* Allow configuration of each element that will be transmitted into the client’s stream.

Devices should have configurable what elements will be sent to the web system.

* 1. **Telemetry Web Front-End**
     1. *[Critical]* Display telemetry data in visual form.

We will have a web front end that will allow the viewing of telemetry data from devices in graphs, charts, etc.

* + 1. *[High]* Allow users to switch between devices they observe.

Since we are designing the web system to dynamically accept telemetry data from devices, users should be able to select which device's telemetry data they would like to view.

* + 1. *[High]* Offer search-able historical telemetry information.

The web system should allow users to search through telemetry data and view stored telemetry data.

* + 1. *[Medium]* Allow users to select sensors to be displayed.

Allow the users to select what sensors are displayed from a device, allowing them to customize the page to their needs and clearing clutter on the page.

* + 1. *[Medium]* Allow users to monitor multiple device sensor information using same view.

Expanding on requirements 2.2.4 this allows the user to view data from two separate devices on the same page, allowing further customization and reducing clutter further.

* + 1. *[Low]* Allow users to switch visual display used for individual sensors.

This would allow the user to make the decision on what type of display used, some examples could be using a chart, graph, or gauge to display velocity information from a device.

* + 1. *[Low]* Provide download-able segments between “MARKS” and/or time periods.

This would allow users to download a segment for analysis without relying on the web system.

* 1. **Telemetry Web Back-End**
     1. *[Critical]* Record telemetry streams from devices.

When receiving data from devices, we will decode and store it in the database.

* + 1. *[High]* Watch multiple telemetry data streams being sent by devices.

Our web system should allow for multiple devices to be streaming data to it while still performing all other functionality listed in our requirements.

* + 1. *[Medium]* Decode video frames from telemetry stream.

Video frames will come encoded so that they may be transmitted via our protocol, once they reach the web system they will need to be decoded before we store them.

* + 1. *[Medium]* Maintain a list of “MARKS” for quick historical referencing.

Marks will be treated differently than normal telemetry data and will be stored in a different database table altogether.

* + 1. *[Low]* Respond to XML response requests.

We would like the system to be able to respond to XML requests and return data to the requesting device. Allowing the web system to retrieve data not readily accessible to the device or perform calculations for the device.

## Non-Functional Requirements

The Non-Functional Requirements will cover the “Who” of our system. This will cover how the system will accomplish its functional requirements and aspects of the project that will affect our functional requirements.

1. **Supportability**

This area covers the non-functional requirements of the system that will affect its future use and compatibility in its deployment platform

* 1. Telemetry web system
     1. The system shall be viewable in the latest version of Mozilla Firefox.

The system should use formatting that is compatible with this browsing platform.

* + 1. The system shall be viewable in the latest version of Google Chrome.

The system should use formatting that is compatible with this browsing platform.

* + 1. The system shall be viewable in the latest version of Microsoft Internet Explorer.

The system should use formatting that is compatible with this browsing platform.

* + 1. The system shall dynamically accept any data stream that conforms to our protocol.

The telemetry web system should be agnostic to what device is sending the telemetry stream or how, it should only be concerned with the protocol it is sent in.

1. **Design Constraints**

These non-functional requirements are any requirement that restrict us to using certain tools, developing certain ways, or which platforms we may release on.

* 1. ECU Embedded Control Unit
     1. The system shall run on an Arduino’s chip set.

The Arduino’s was picked for its low cost and large set of feature. Having the larger number of features at our disposal makes sure that it meets current and future requirements.

* + 1. The system shall be unobtrusive to the user of the Velomobile.

The system must be unobtrusive to the user, allowing them to focus on the operation of the Velomobile.

1. **Web System**
   1. The system shall run on Windows Server.

Windows server allows us to work in the ASP.NET/C# environment which we are most familiar with, thus improving performance.

* 1. The system shall use Microsoft SQL Server as its database.

Due to its ease of use and accessibility to the team, it was decided this would be the database server that would serve our needs best.

## Requirements Trace Matrix

Table 3.3: Requirements Trace Matrix, shows the relationships between the Functional Requirements of the system and the Use Cases involved.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Requirements | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Use Case |  | 1.1.1 | 1.1.2 | 1.1.3 | 1.1.4 | 1.1.5 | 1.2.1 | 1.2.2 | 1.2.3 | 1.3.1 | 1.3.2 | 1.3.3 | 1.4.1 | 1.4.2 | 2.1.1 | 2.1.2 | 2.1.3 | 2.1.4 | 2.1.5 | 2.1.6 | 2.2.1 | 2.2.2 | 2.2.3 | 2.2.4 | 2.2.5 | 2.2.6 | 2.2.7 | 2.3.1 | 2.3.2 | 2.3.3 | 2.3.4 | 2.3.5 |
| Change Power Assist Level |  | X | X |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Change Speed |  |  |  |  |  | X | X |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Read Sensors | X | X | X | X | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Display Data |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Display Velocity |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Pair Bluetooth Device |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Transmit Telemetry Stream |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Parse Telemetry Stream |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Display Telemetry Data |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X |  |  |  |  | X |  |  |  |  |
| Store Telemetry Data |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X |  |  |  |  | X |  |  |  |  |
| Retrieve Telemetry Data |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X |  |  |  |  | X |  |  |  |  |
| Start Up |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Shut Down |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Table 3.3: Requirements Trace Matrix

## Purchased Components

Table 3.4 covers the components which have been selected for our project according to various system requirements.

|  |  |  |
| --- | --- | --- |
| Product | # | Cost |
| NiMH 24V 10000mAh Battery | 1 | 149.99 |
| 19.2V-24V Smart NiMH Battery Charger | 1 | 50.46 |
| TouchShield Slide | 1 | 174.93 |
| Bluetooth Module BlueSMiRF Gold | 1 | 64.95 |
| RS232 Shifter SMD | 1 | 13.95 |
| AttoPilot Voltage and Current Sense | 1 | 19.95 |
| UART JPEG Color Camera | 1 | 54.95 |
| Logic Level Converter | 2 | 1.95 |
| Hall Effect Sensor | 2 | 0.95 |
| Real Time Clock Module | 1 | 19.95 |
| Total |  | $ 552.03 |

**Table 3.4: Price Guide for all Components**

## Interfaces

Interfaces specify the forms and methods of communication that are supported in the application.

### Rider Interfaces

The current ECU User interfaces, shown in figures 3.5.1a and 3.5.1b, allows the user to switch between screens of operation to access additional settings and information, or add/subtract power assist. The border remains the same as the other aspects of the screen vary, allowing the user easy navigation between screens at all times. Figure 3.5.1a is an example of the primary interface displayed while the velomobile is in use, providing the rider with a simple current speed display. Figure 3.5.1b represents the secondary display, informing the rider of the distance they have traveled on their current ride, as well as the ride’s duration.

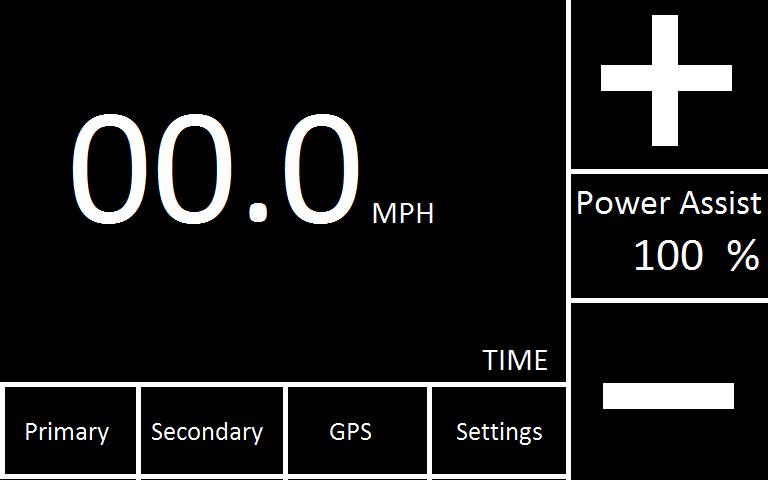
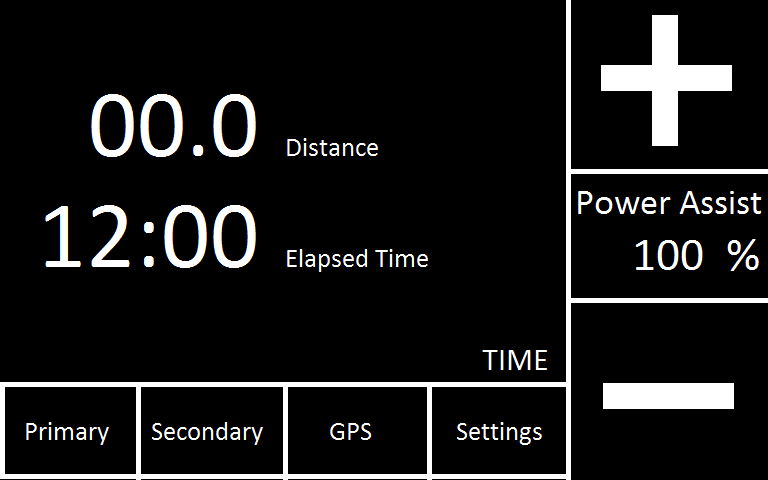


Figure .5.1a: ECU Primary Interface



**Figure 3.5.1b: ECU Secondary Interface**

#### Navigation

The navigation elements on the ECU will be large enough to accommodate a driving glove and easily visible from the riders seat to a rider with normal vision. Navigation will show the information relative to the page and any key information that should be always visible.

#### Safety

In any moving vehicle safety for the rider and those around is a concern that must be addressed. For this reason the navigation to configuration screens will be disabled with the Velomobile is moving at a rate above 1 mph.

### Web Interfaces

The Telemetry Web System, shown in figure 3.5.2, allows users to view the metrics that have been broadcast from a device. The UI will allow the user to quickly switch between live and historical data. Also allowing the user to switch to the other devices that have broadcast there metrics to our system.

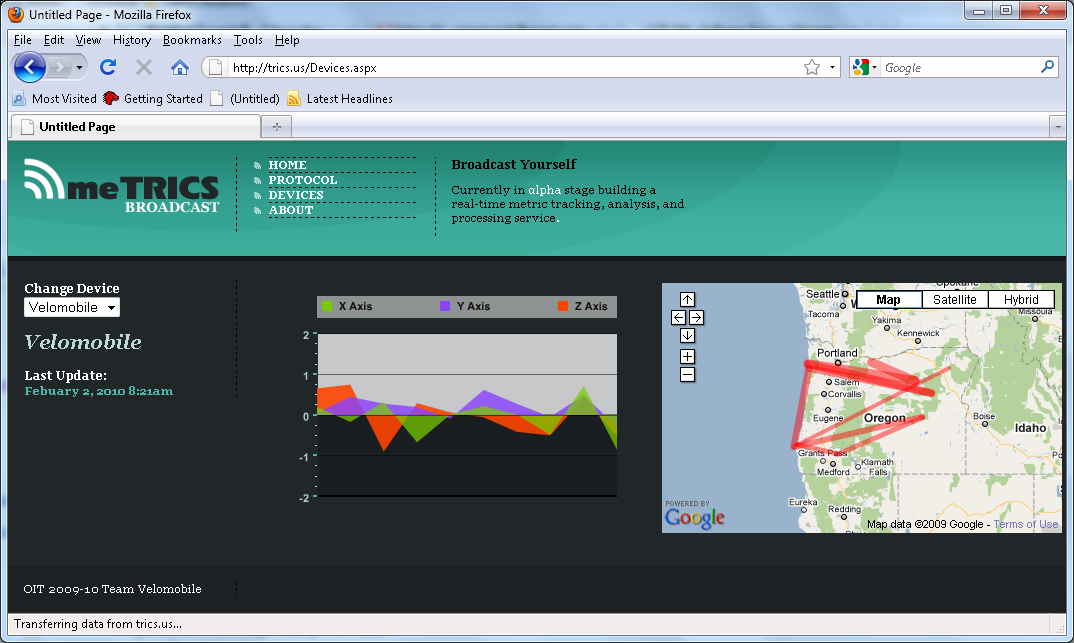


Figure 3.5.2: Telemetry Web System

### Hardware Interfaces

This section defines the physical reactions that will happen between the ECU and its internal components.

#### RS232 Requirements

Recommended Standard 232 will be used to interface devices inside the ECU.

##### General Processing Flow

The ECU will read one of four RS232 ports and report the gathered data to the device library function outlined. {[?] This information will be gathered during a standard processing loop / This information will be gathered at the time of receipt.}

##### Interface Level Requirements

Additional hardware will also be required if the control voltage level is to be anything other than 5 Volts.

#### PWM Requirements

A standard control signal for the electric motor controllers is Pulse Width Modulation. This control signal will be available from the ECU.

##### General Processing Steps

The ECU will evaluate the current circumstance of the Velomobile’s motion and provide a PWM control signal to the motor controller over the PWM interface.

##### Interface Processing Time Requirements

As outlined in Interface Overview > Constraints the PWM frequency must fall between 30.6373 Hertz and 31372.5 Hertz.

##### Interface Level Requirements

Additional hardware will also be required if the control voltage level is to be anything other than 5 Volts.

#### Analog Voltage Level Requirements

A standard control signal for the electric motor controllers is an Analog Voltage Level. This control signal will be available from the ECU.

##### General Processing Steps

The ECU will evaluate the current circumstance of the Velomobile’s motion and provide an analog voltage control signal to the motor controller over the analog interface.

##### Interface Level Requirements

The voltage level from the analog signal will be between 0V and 5V. Additional hardware will also be required if the control voltage level is required to be anything other than 5 Volts. The analog voltage will be available with 256 levels or an 8 bit resolution allowing for changes of [A=5V/256] or 19.5 mV.

### Software Interfaces

There are currently no software interfaces being used to interface components of the software system.

### Communication Interfaces

The communication interface that will be used to talk between the Velomobile and the Telemetry Web System, XML-meTRICS is outlined below.

#### XML-meTRICS Specification

This is the specifications of the XML-meTRICS protocol. The information below identifies the structure and implementation of the protocol.

##### Request

The following represents the processing of the data by the website; the content type will inform the website what is to be done with the information that follows.

POST HTTP  
Host: trics.us  
Content-Type: metric/xml

POST HTTP  
Host: trics.us  
Content-Type: metric/image

##### Stream (metric/xml)

The parser incorporated into the website will receive xml posts from devices such as the velomobile. The following is an example of the format expected by the website.

[Fields]  
{Optional}  
{<?xml version="1.0" encoding="[SEE DATATYPES: Encoding]" ? >}  
<[DEVICE] {{{t}|{timestamp}="[TIMESTAMP]"}|{{d}|{datetime}="[DATETIME]"}}>  
        <[ELEMENT]{{{t}|{timestamp}="[TIMESTAMP]"}|{{d}|{datetime}="[DATETIME]"}} {{p|process}="[SEE PROCESSING]"}>[DATA]</[ELEMENT]>  
        <[ELEMENT]...</[ELEMENT]>  
</[DEVICE]>

##### Stream (metric/image)

The web system will be able to interpret binary images sent over the web. These images may be decrypted from their data format into viewable jpeg style pictures.

##### Data Types

The types of information

* Boolean Types:
  + True | true
  + False | false
* Coordinates Types:
  + DMS – Degrees : Minutes : Seconds (49°30'00", 123°30'00"W)
  + DM – Degrees : Decimal Minutes (49°30.0', -123°30.0'), (49d30.0m,-123d30.0')
  + DD - Decimal Degrees (49.5000,-123.5000)
* Encoding Types:
  + UTF-8
  + UTF-18
  + ISO-8859-1
* Data  
  Stipulations: Requires that the xml stream header allow data type.
  + Types:
    - Base64
    - ASCII
    - UTF-8

#### Displays

Some of the displays available for end users of the website will include:

* Google Maps
* Accelerometer
* Compass

#### Processing

Processing of information sent to the website from a device such as the velomobile will include:

* Numerical.Bounds
* Numerical.Average
* Mapping.Path.Distance
* Mapping.Path.Route
* Twitter.Post
* [Device Name].[Element ID]

# Use Case Model Survey

The purpose of this section is to provide a brief description of the project and to describe the known actors and how they interact with the system. This document is a high level view and more information can be found in the individual use case documents.

## Actors

The actors involved with the system’s ECU side are all traced back to interrupts. These Interrupts are each involved with the various use cases of the ECU. Figure 4.1 shows the inheritance of the interrupt based actors.

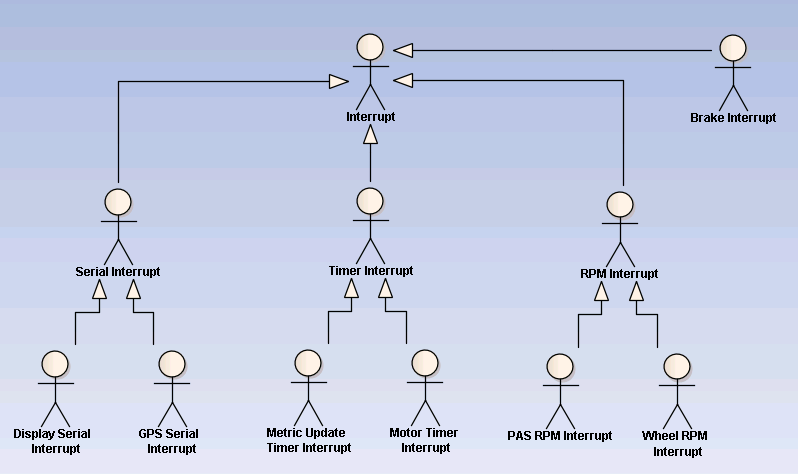
****

Figure 4.1: ECU Actor Hierarchy

### Interrupt

The base interrupt all sensors are tied into.

#### Brake Interrupt

A brake interrupt will be triggered when the brakes of the velomobile are used.

#### Serial Interrupt

A generalized serial input interrupt.

#### Display Serial Interrupt

Derived from the Serial Interrupt, this interrupt actor facilitates communication between the screen and the ECU over a UART connection.

#### GPS Serial Interrupt

Derived from the Serial Interrupt, this UART interrupt will allow reception of GPS NEMA strings.

#### Timer Interrupt

A generalized timer interrupt.

#### Metric Update Timer Interrupt

Derived from the timer interrupt, this interrupt lives in the processor of the ECU. It prompts the system for updates to sensor readings.

#### Motor Timer Interrupt

Derived from the timer interrupt, this interrupt lives in the processor of the ECU. It maintains the power control signals being sent to the motor.

#### RPM Interrupt

A generalized RPM interrupt.

#### PAS RPM Interrupt

Derived from the RPM interrupt, monitors the pulse signal from the PAS, checking for the speed of rotation.

#### Wheel RPM Interrupt

Derived from the RPM interrupt, checks the pulse signal from the drive wheel, checking the speed of rotation.

### Rider

The field user is involved in the system via uploading data to the web. They have access to setting marks in the data they upload, and all data uploaded depends on their control of the Velomobile.

### Web User

The web user is involved in the system via the website. They will be able to search for data according to various factors (user, date, marks).

### Web Viewer

The web viewer will be able to view the website at anytime, view any incoming data, and sort the data anyway they wish.

### AJAX Timer

The AJAX timer runs on client’s computer, and will alert the web server when new data information is required while viewing streaming or pre-existing data.

### Telemetry Device

The Arduino will query the Bluetooth device for the latest information it has received from the modem.

## Use Case Modeling

The following figures (4.2 and 4.3) display the use cases of the ECU and the web system as well as their actor relationships. Due to the real time operating environment of the ECU, actors which modify the states within the ECU are primarily interrupts. Most interrupts stem from sensors on various parts of the velomobile such as the wheel RPM sensor which reads revolutions of the drive wheel.

The actors of the web system on the other hand are either human or other parts of the system. Web users may select what data to display as well as how it is represented. Also, the ECU or comparable devices sending data to the parser invoke change in the parser, which in turn sends the storable data to the database.

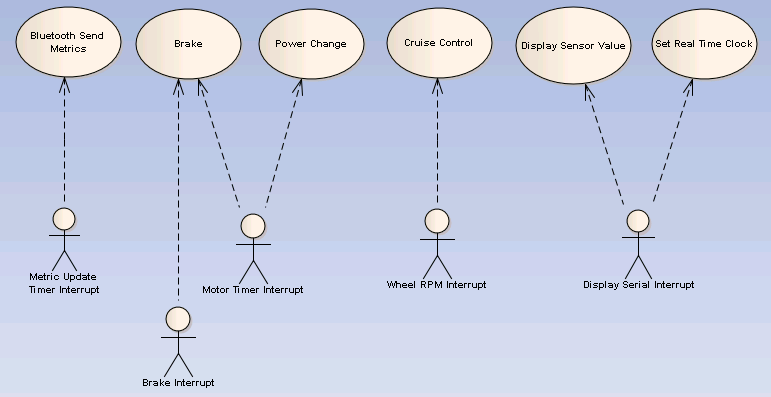


Figure 4.2: ECU Use Case Model

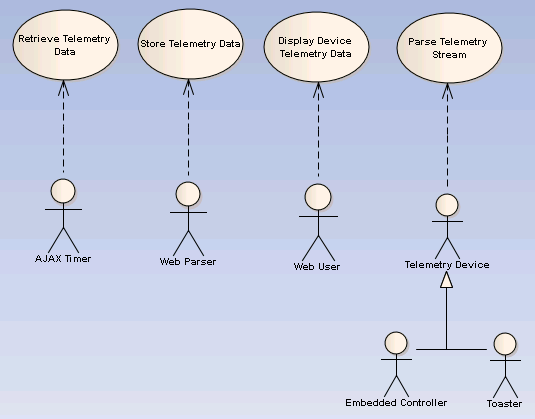


Figure 4.3: Web Use Case Model

# System Architecture

## Functional Architecture

Figure 5.1: Functional Architecture illustrates our system as it is broken down into its functional packages. The data stream encapsulates the interaction of devices and the web system, and also how the ECU will interact with the web system. This covers the XML stream and parsing.

On the ECU side we have quite a few packages that relate directly to it. Sensor input contains the requirements for the ECU’s sensors and how they are stored and retrieved. Calculations relates to requirements for power assist calculations mostly, along with other miscellaneous calculations that may need to be performed. Power assist control package will rely on the calculations package for sensor information to use when providing power assist to the Velomobile. Embedded display relates to the ECU and its attached display’s requirements.

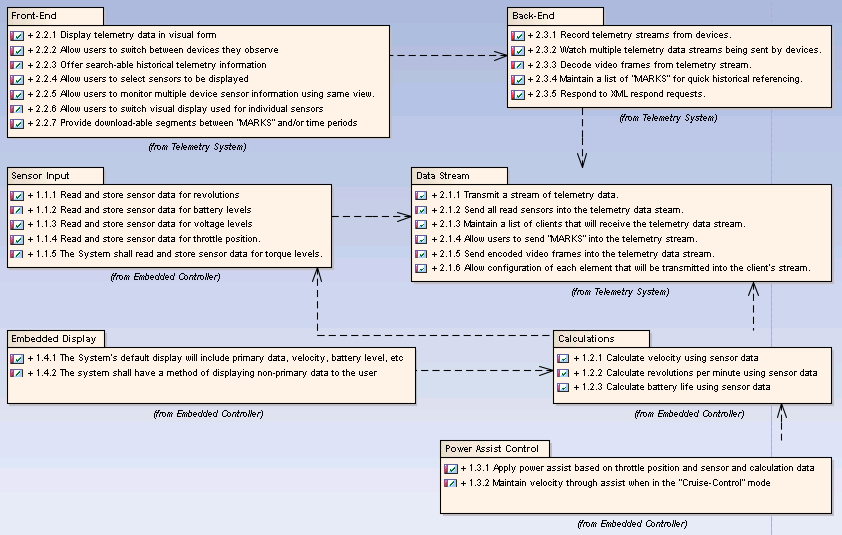
Over on the web-server side we have backend which deals for the most part with our persistence layer and database interaction. Storage and retrieval are going to be critical requirements for our system. The front end package deals with the display of the website and telemetry data via graphs and charts. 

Figure 5.1: Functional Architecture

Back End (Web): Relies on the data stream information.

Calculations: Relies on sensor input and the data stream.

Data Stream: Relies on values returned from all sensors.

Embedded Display: Relies on return values of all calculations.

Front End (Web): Relies on the data processed and stored by the back end (web).

Power Assist Control: Relies on return values of all calculations.

Sensor Input: Reads in all sensor values.

## Non-Functional Architecture

Figure 5.2 displays the packages of our non-functional requirements. Our functional architecture covers key non functional requirements of our system like what web browsers will be able to access the system, or what ECU our system will operate on.



Figure 5.2: Non-Functional Architecture

## Physical Architecture

Due to the complexity of this system it has been broken down into two physical layers.

The first layer is the communication layer shown in Figure 5.3a: Physical Communication Layer depicting how the systems talk to each other. The additional controller ( with the image of a toaster ) was put in this diagram to emphasize the fact that our web system will track telemetry data for any device, as long as it is registered with the web system and is broadcasting some form of telemetry data in our protocol. The web server is the home of our web system and our database, storing all telemetry data and providing it to clients who request it. The Bluetooth modem is the internet interface we shall be using between the ECU for transmitting telemetry data from the ECU and returning acknowledgements to the ECU.

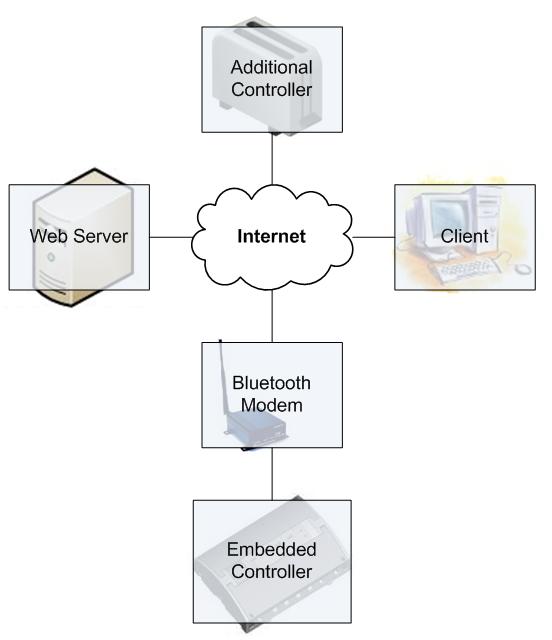


Figure 5.3a: Physical Communication Layer

Next we show the component layer in . This shows each relationship between the ECU’s components. Both the ECU and the display will be Arduino Embedded Controllers. All logic will run on the ECU except for some of the display which will be handled by the display. Using interrupt driven code the ECU will transmit display information to the display, which will handle all display of information and return interrupts to the ECU when the user interacts with the display. The AttoPilot is a pivotal piece of our system which allows us to measure Voltage and Current into the motor controller, allowing us to calculate an estimate of the power output of the motor. The Bluetooth component is an important component for our telemetry streaming. It will connect to the Bluetooth modem, providing an internet connection to use when streaming telemetry data to the web system. The motor controller is the intermediary between the ECU and the motor that provides power assist.

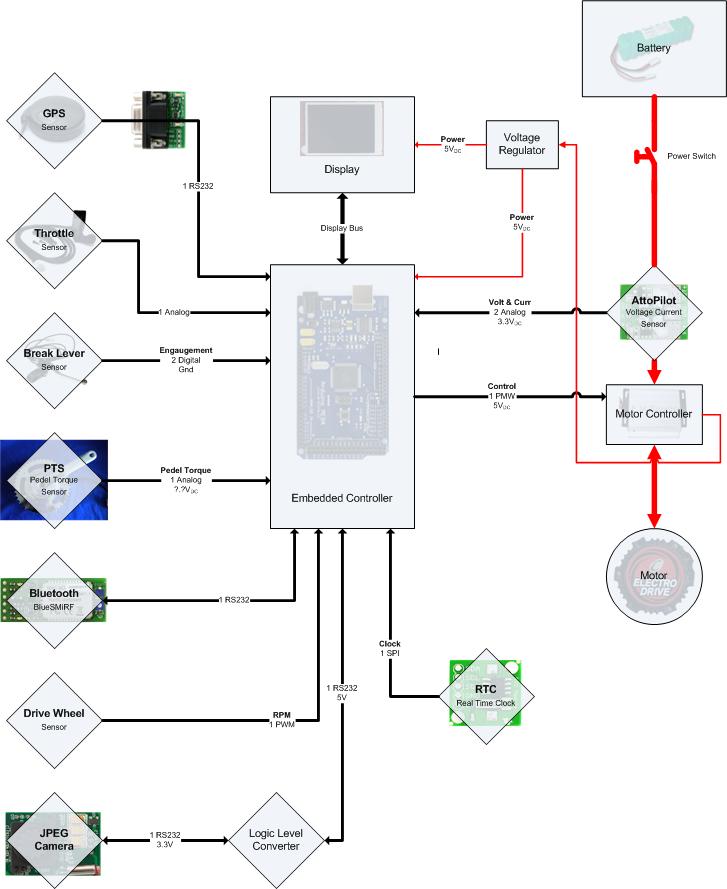


Figure 5.3b: Physical Component Layer

AttoPilot: Reads the current and voltage output levels.

Battery: Powers the system.

Bluetooth: Sends data to the website via modem.

Break Lever: Senses when breaks are applied, as to cut power to the motor.

Camera: JPEG Camera.

Clock: Real time clock.

Drive Wheel: Senses RPM.

ECU: Runs all calculations, sends signals to display and Bluetooth.

GPS: Reads GPS information and sends it to the ECU.

Motor: Brushless high RPM electric motor.

Motor Controller: Sends power to the motor.

Pedal Torque Sensor: Determines power input from the rider.

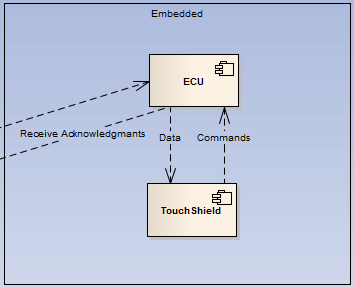
Throttle: Determines power feed to the motor.

Voltage Regulator: Steps the power up or down from the battery to the motor.

## Component Architecture

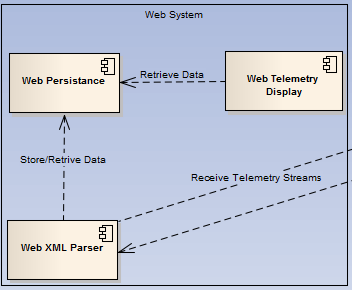
The overlying component architecture defines the correlation of each physical entity in each of the two aspects of the system. The first aspect is the embedded system, as displayed by Figure 5.5. The second aspect is the web system, displayed in Figure 5.6.

Figure 5.4a, the embedded component architecture defines the physical elements of the embedded system and the flow of information between them. The ECU runs all calculations and generates the display that is sent to the Touch Shield. The Touch Shield is the physical screen which displays information to the rider and provides as the rider’s input device to control the system.



**Figure 5.4a: Embedded component architecture**

Figure 5.4b, the web component architecture defines the correlation of the main aspects of the web system. The three main portions are the Telemetry Display, the XML Parser, and the Persistence layer. The telemetry display generates graphs and charts to display data at the web user’s discretion. The XML parser converts incoming data to the proper format in which the persistence layer can store it.



**Figure 5.4b: Web component architecture**

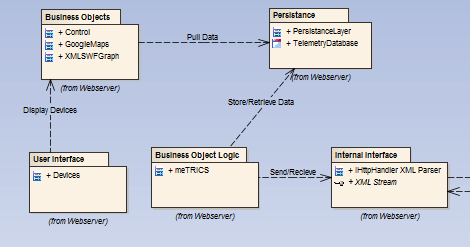
## Logical Architecture

The logical architecture defines the three tiered architecture in place in our system. Each portion of the system has been divided into one of the four categories of the persistence layer, the business objects, the business object logic, and the user interface. Figures 5.5a and 5.5b are linked as one large figure via the connections exiting figure 5.5a on the right, these connections tie into the left of figure 5.5b.

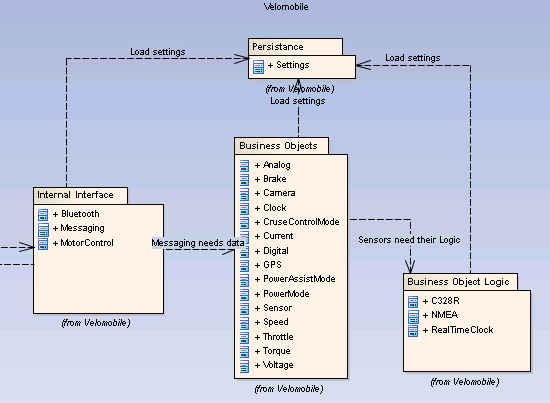
The persistence layer is comprised of the web server’s telemetry database and the settings preloaded in ECU for each sensor on the velomobile.

The business object layer is comprised of the controls related to the graphs/charts on the website, the link to Google maps, the XMLSWF graph, and the wide array of sensors in place on the velomobile which feed data to the ECU.

The business object logic includes the meTRICS website, and the different libraries used on the ECU, as well as the real time clock incorporated on the Arduino.



**Figure 5.5a: Logical Web Server Architecture**



**Figure 5.5b: Logical Velomobile Architecture**