

Clarkson University Intelligent Cars IV

Design Document

Revision A - October 16, 2017

Written by Team KVM:

James Bruska, Milton Griffin, Robin Kuhns,

Timothy Law, Ryan Shippee, Matthew Strenk

**Table of Contents**

[**Scope**](#_gjdgxs) **7**

[Identification](#_30j0zll) 7

[Standards](#_1fob9te) 7

[Definitions](#_3znysh7) 7

[Document Overview](#_2et92p0) 8

[System Operation Overview](#_tyjcwt) 8

[System Component Overview](#_3dy6vkm) 9

[**High Level Design**](#_1t3h5sf) **10**

[Software High Level Design](#_4d34og8) 10

[Graphical User Interface High Level Design](#_2s8eyo1) 11

[Hardware High Level Diagram](#_17dp8vu) 12

[**Software Detailed Design**](#_26in1rg) **13**

[Component Diagrams](#_lnxbz9) 13

[Main](#_35nkun2) 13

[Accuracy Mode](#_1ksv4uv) 14

[Speed Mode](#_44sinio) 15

[Turning Mechanisms](#_2jxsxqh) 16

[Zybo Components](#_z337ya) 16

[Visual Sensor Driver](#_3j2qqm3) 16

[Functional Purpose](#_1y810tw) 16

[Functional Specification](#_4i7ojhp) 16

[Component Function Designs](#_1ci93xb) 17

[Component Variable Designs](#_3whwml4) 18

[Hardware Connections](#_2bn6wsx) 18

[Pan-Tilt Servo Driver [Design Phase 2]](#_2p2csry) 19

[Functional Purpose](#_147n2zr) 19

[Functional Specification](#_3o7alnk) 19

[Component Function Designs](#_23ckvvd) 19

[Component Variable Designs](#_ihv636) 19

[Internal Function Designs](#_32hioqz) 19

[Internal Variable Designs](#_1hmsyys) 19

[Hardware Connections](#_41mghml) 19

[Linescan Camera Driver](#_2grqrue) 19

[Functional Purpose](#_vx1227) 19

[Functional Specification](#_3fwokq0) 19

[Component Function Designs](#_1v1yuxt) 21

[Component Variable Designs](#_4f1mdlm) 22

[Hardware Connections](#_2u6wntf) 22

[Motor Driver](#_19c6y18) 23

[Functional Purpose](#_3tbugp1) 23

[Functional Specification](#_28h4qwu) 23

[Component Function Designs](#_nmf14n) 23

[Component Variable Designs](#_37m2jsg) 23

[Hardware Connections](#_1mrcu09) 24

[Steering Servo Driver](#_46r0co2) 24

[Functional Purpose](#_2lwamvv) 24

[Functional Specification](#_111kx3o) 24

[Component Function Designs](#_3l18frh) 25

[Component Variable Designs](#_206ipza) 25

[Hardware Connections](#_4k668n3) 25

[LIDAR Driver [Design Phase 2]](#_2zbgiuw) 25

[Functional Purpose](#_1egqt2p) 25

[Functional Specification](#_3ygebqi) 25

[Component Function Designs](#_2dlolyb) 25

[Component Variable Designs](#_sqyw64) 25

[Internal Function Designs](#_3cqmetx) 26

[Internal Variable Designs](#_1rvwp1q) 26

[Hardware Connections](#_4bvk7pj) 26

[Bluetooth Driver [Design Phase 2]](#_2r0uhxc) 26

[Functional Purpose](#_1664s55) 26

[Functional Specification](#_3q5sasy) 26

[Component Function Designs](#_25b2l0r) 26

[Component Variable Designs](#_kgcv8k) 26

[Internal Function Designs](#_34g0dwd) 26

[Internal Variable Designs](#_1jlao46) 26

[Hardware Connections](#_43ky6rz) 26

[FRDM Components](#_2iq8gzs) 26

[Visual Sensor Driver](#_xvir7l) 26

[Functional Purpose](#_3hv69ve) 26

[Functional Specification](#_1x0gk37) 26

[Component Function Designs](#_2w5ecyt) 27

[Component Variable Designs](#_1baon6m) 28

[Hardware Connections](#_3vac5uf) 28

[Pan-Tilt Servo Driver [Design Phase 2]](#_pkwqa1) 28

[Functional Purpose](#_39kk8xu) 28

[Functional Specification](#_1opuj5n) 28

[Component Function Designs](#_48pi1tg) 29

[Component Variable Designs](#_2nusc19) 29

[Internal Function Designs](#_1302m92) 29

[Internal Variable Designs](#_3mzq4wv) 29

[Hardware Connections](#_2250f4o) 29

[Line Scan Camera Driver](#_haapch) 29

[Functional Purpose](#_319y80a) 29

[Functional Specification](#_1gf8i83) 29

[Component Function Designs](#_40ew0vw) 31

[Component Variable Designs](#_2fk6b3p) 32

[Hardware Connections](#_upglbi) 32

[Motor Driver](#_3ep43zb) 33

[Functional Purpose](#_1tuee74) 33

[Functional Specification](#_4du1wux) 33

[Component Function Designs](#_2szc72q) 33

[Component Variable Designs](#_184mhaj) 33

[Hardware Connections](#_3s49zyc) 34

[Steering Servo Driver](#_279ka65) 34

[Functional Purpose](#_meukdy) 34

[Functional Specification](#_36ei31r) 34

[Component Function Designs](#_1ljsd9k) 35

[Component Variable Designs](#_45jfvxd) 35

[Hardware Connections](#_2koq656) 35

[LIDAR Driver [Design Phase 2]](#_zu0gcz) 35

[Functional Purpose](#_3jtnz0s) 35

[Functional Specification](#_1yyy98l) 35

[Component Function Designs](#_4iylrwe) 35

[Component Variable Designs](#_2y3w247) 35

[Internal Function Designs](#_1d96cc0) 36

[Internal Variable Designs](#_3x8tuzt) 36

[Hardware Connections](#_2ce457m) 36

[Bluetooth Driver [Design Phase 2]](#_rjefff) 36

[Functional Purpose](#_3bj1y38) 36

[Functional Specification](#_1qoc8b1) 36

[Component Function Designs](#_4anzqyu) 36

[Component Variable Designs](#_2pta16n) 36

[Internal Function Designs](#_14ykbeg) 36

[Internal Variable Designs](#_3oy7u29) 36

[Hardware Connections](#_243i4a2) 36

[Generic Components](#_j8sehv) 36

[Line Following Mechanism](#_338fx5o) 36

[Functional Purpose](#_1idq7dh) 36

[Functional Specification](#_42ddq1a) 36

[Component Function Designs](#_2hio093) 38

[Component Variable Designs](#_wnyagw) 38

[Turning Mechanism](#_3gnlt4p) 39

[Functional Purpose](#_1vsw3ci) 39

[Functional Specification](#_4fsjm0b) 39

[Component Function Designs](#_2uxtw84) 39

[Component Variable Designs](#_1a346fx) 41

[Accuracy Mode Logic](#_3u2rp3q) 41

[Functional Purpose](#_2981zbj) 41

[Functional Specification](#_odc9jc) 41

[Component Function Designs](#_2mn7vak) 43

[Component Variable Designs](#_11si5id) 44

[Speed Mode Logic](#_3ls5o66) 44

[Functional Purpose](#_20xfydz) 44

[Functional Specification](#_4kx3h1s) 44

[Component Variable Designs](#_302dr9l) 46

[Discovery Mode Logic [Design Phase 2]](#_1f7o1he) 46

[Functional Purpose](#_3z7bk57) 46

[Functional Specification](#_2eclud0) 46

[Component Function Designs](#_thw4kt) 46

[Component Variable Designs](#_3dhjn8m) 46

[Internal Function Designs](#_1smtxgf) 46

[Internal Variable Designs](#_4cmhg48) 46

[Collision Mode Logic [Design Phase 2]](#_2rrrqc1) 46

[Functional Purpose](#_16x20ju) 46

[Functional Specification](#_3qwpj7n) 46

[Component Function Designs](#_261ztfg) 47

[Component Variable Designs](#_l7a3n9) 47

[Internal Function Designs](#_356xmb2) 47

[Internal Variable Designs](#_1kc7wiv) 47

[Manual Mode Logic [Design Phase 2]](#_44bvf6o) 47

[Functional Purpose](#_2jh5peh) 47

[Functional Specification](#_ymfzma) 47

[Component Function Designs](#_3im3ia3) 47

[Component Variable Designs](#_1xrdshw) 47

[Internal Function Designs](#_4hr1b5p) 47

[Internal Variable Designs](#_2wwbldi) 47

[Graphical User Interface](#_1c1lvlb) 47

[Functional Purpose](#_3w19e94) 47

[Functional Specification](#_2b6jogx) 47

[Interface Signals and variables specification](#_qbtyoq) 48

[Data items needed](#_3abhhcj) 48

[Pseudo Code](#_13qzunr) 49

[**Hardware Detailed Design**](#_i17xr6) **52**

[Zybo Micro-Controller](#_320vgez) 52

[Part Name](#_1h65qms) 52

[Functional Specification](#_415t9al) 52

[Schematics](#_2gb3jie) 52

[Signal Information](#_vgdtq7) 54

[Mechanical Designs](#_3fg1ce0) 56

[Acrylic Backplate](#_1ulbmlt) 56

[Module Function Specification](#_4ekz59m) 57

[FRDM Micro-Controller](#_2tq9fhf) 57

[Part Name](#_18vjpp8) 57

[Functional Specification](#_3sv78d1) 57

[Schematics](#_280hiku) 57

[Signal Information](#_n5rssn) 59

[Mechanical Designs](#_375fbgg) 60

[Acrylic Backplate](#_1maplo9) 60

[Module Function Specification](#_2lfnejv) 60

[Visual Sensor](#_10kxoro) 60

[Part Name](#_3kkl7fh) 60

[Functional Specification](#_1zpvhna) 60

[Schematics](#_4jpj0b3) 60

[Signal Information](#_2yutaiw) 62

[Mechanical Designs](#_1e03kqp) 62

[Module Function Specification](#_3xzr3ei) 62

[Line Scan Camera](#_2d51dmb) 62

[Part Name](#_sabnu4) 62

[Functional Specification](#_3c9z6hx) 63

[Schematics](#_1rf9gpq) 63

[Signal Information](#_24ufcor) 64

[Mechanical Designs](#_33zd5kd) 64

[Module Function Specification](#_3hej1je) 66

[H-Bridge](#_4gjguf0) 66

[Part Name](#_2vor4mt) 66

[Functional Specification](#_1au1eum) 67

[Schematics](#_3utoxif) 67

[Signal Information](#_1o97atn) 68

[Mechanical Designs](#_488uthg) 68

[Module Function Specification](#_21od6so) 69

[Voltage Regulator Module](#_gtnh0h) 69

[Part Name](#_30tazoa) 69

[Functional Specification](#_1fyl9w3) 69

[Schematics](#_3zy8sjw) 69

[Signal Information](#_1tdr5v4) 69

[Mechanical Designs [Design Phase 2]](#_4ddeoix) 70

[Module Function Specification](#_2sioyqq) 70

[LIDAR [Design Phase 2]](#_17nz8yj) 70

[Part Name](#_3rnmrmc) 70

[Functional Specification](#_26sx1u5) 70

[Schematics](#_ly7c1y) 70

[Signal Information](#_35xuupr) 70

[Mechanical Designs](#_1l354xk) 70

[Module Function Specification](#_452snld) 70

[Bluetooth [Design Phase 2]](#_2k82xt6) 70

[Part Name](#_zdd80z) 70

[Functional Specification](#_3jd0qos) 70

[Schematics](#_1yib0wl) 70

[Signal Information](#_4ihyjke) 70

[Mechanical Designs](#_2xn8ts7) 71

[Module Function Specification](#_1csj400) 71

[**Mechanical Detailed Design**](#_3ws6mnt) **71**

[Full Car Design](#_2bxgwvm) 71

[**References**](#_r2r73f) **72**

# Scope

## Identification

This document serves as the Detailed Design Document for the Clarkson University Intelligent Cars IV project as part of the University’s Computer Engineering Senior Lab class.

## Standards

This design follows the standards approved in the IEEE 1012-2012 standard for system and software verification and validation. This design follows the standards approved in the IEEE 802.15.4 standard for wireless personal area networks. This design follows the standards approved in the IEEE 113-1985 standard for test procedures for direct-current machines.

## Definitions

* + 1. Car: either one of the vehicles in production which carries either of the main controllers
    2. Track: a 2.54cm wide black curve on a white background
    3. Track piece: an object with the track on it
    4. Front: When used in respect to the car, it is a designated side of the car
    5. Forward: When used in respect to the car, it is a singular directional vector based on a single designated point and extends away from the car perpendicular to the front
    6. Vision sensor: the CMUcam5 Pixy camera
    7. Directional marker: a length of PVC, standing on one of its non-rounded sides, with two pieces of distinct colored duct tape wrapped around the side furthest from the ground
    8. Placed within: When used in respect to the car’s position, it means that the car position, in relation to the object it is being placed near, is measured from the part of the external object nearest to the car to the part of the car nearest to the external object
    9. Turn around: When used in respect to the movement of the car, it means that the car shall perform a maneuver such that be able to travel forward in a direction 180 degrees from the direction it was traveling prior
    10. Stop: When used in respect to the movement of the car, it means that the car shall cease movement
    11. Continue: When used in respect to the movement of the car, it means that the car shall continue moving forward along the track
    12. Main controller: A reference to either one of the FRDM-K64F board or the Zynq-based board
    13. Signature X: a directional marker in which the two pieces of colored duct tape are different colors in order to indicate a unique combinational signature, ‘X’
    14. Power up: Describes the event in which the car is turned on
    15. Starting line: a point on the track that has a another perpendicular line on each side of the track
    16. Around the track: When used in respect to the car’s movement, this means that the car will traverse the track while either keeping at least two wheels on the track pieces during speed mode or keeping the track between the wheels during all other modes
    17. Separation between cars: The shortest linear distance between the two measured cars
    18. Other computer: A computer that is not the main controller
    19. External interaction: Interaction between the main controller or any other part of the car with any other computer, human, or car

## Document Overview

This document outlines the detailed design for the autonomous car designed the KVM development team. All block diagrams and component descriptions are subject to change based on the needs of the development team. Enjoy.

## System Operation Overview

The system under development which is referenced by this document (Clarkson University Intelligent Cars IV) is a project undertaken by the participating members of the Senior Lab class of the Computer Engineering program. The project task is to develop, design, build, and test a functional intelligent small-scale vehicle which has the ability to perform several different modes of operation. The underlying basis of each of these modes is a small track which the car must navigate in some fashion depending on the mode. These modes are outlined in the Product Concept in the Supporting Documents section of this document.

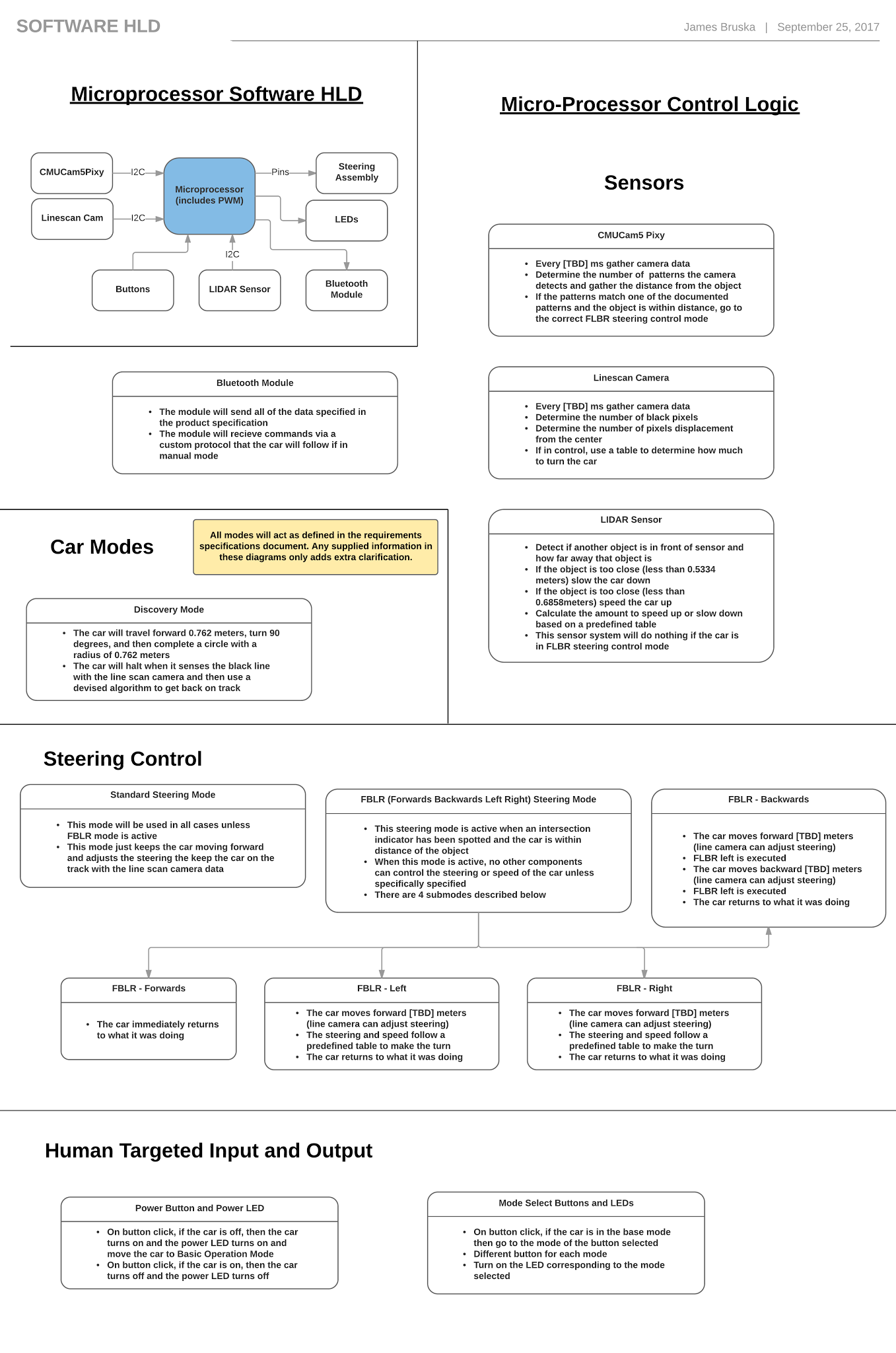
The vehicle itself is to be comprised of a remote control car chassis with attached dual-drive DC motors to power the wheels. The vehicle will also have a servo steering mechanism, a CMUcam5 Pixy camera to recognize directional markers on the track, a line scanner to recognize the track itself, and a pan-tilt mechanism to alter the viewpoint of the Pixy camera. There will be two separate boards which will each have independently the task of serving as the central processing unit of the vehicle itself at different times (each capable of performing the same tasks and fulfilling the outlined requirements). These boards will have the ability to be swapped with minimal modification to function properly. Each board shall be able to direct the car in every task outlined below and will be programmed as such.

## System Component Overview

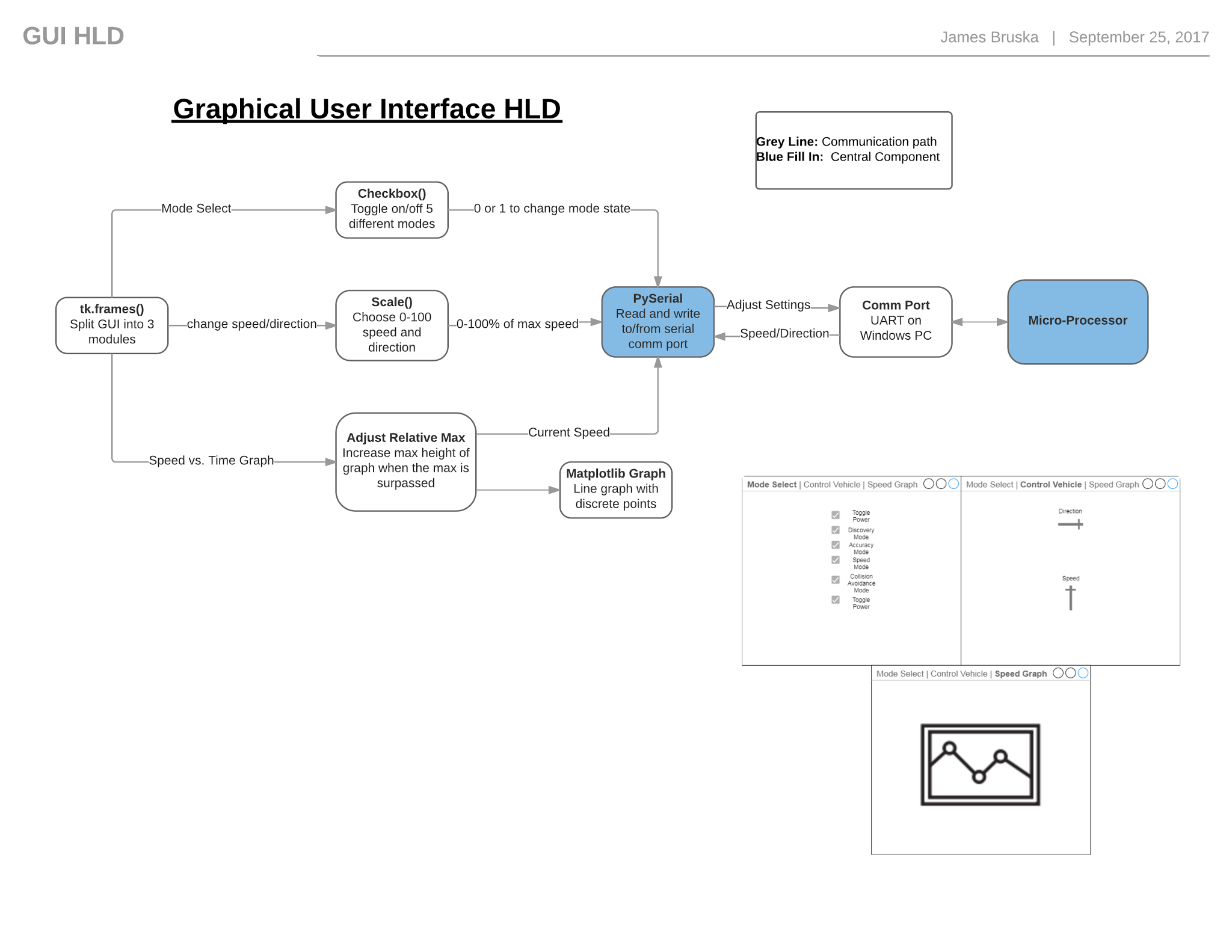
The system is comprised of eight(8) primary physical components and two(2) primary software components. These components consist of: two(2) controller boards, two(2) cameras, one(1) motor with driver board, one(1) steering servo, one(1) LIDAR sensor, one(1) bluetooth communication device, communication protocols, and a web based user interface. In any one(1) instance of the autonomous vehicle, only one controller board may be present. The cameras and LIDAR sensor provide information to the system regarding the current environment. The motor and steering components react based on the information received about the environment. The bluetooth communication device, communication protocols, and web based user interface interact to form a diagnostic display on a separate computer.

# High Level Design

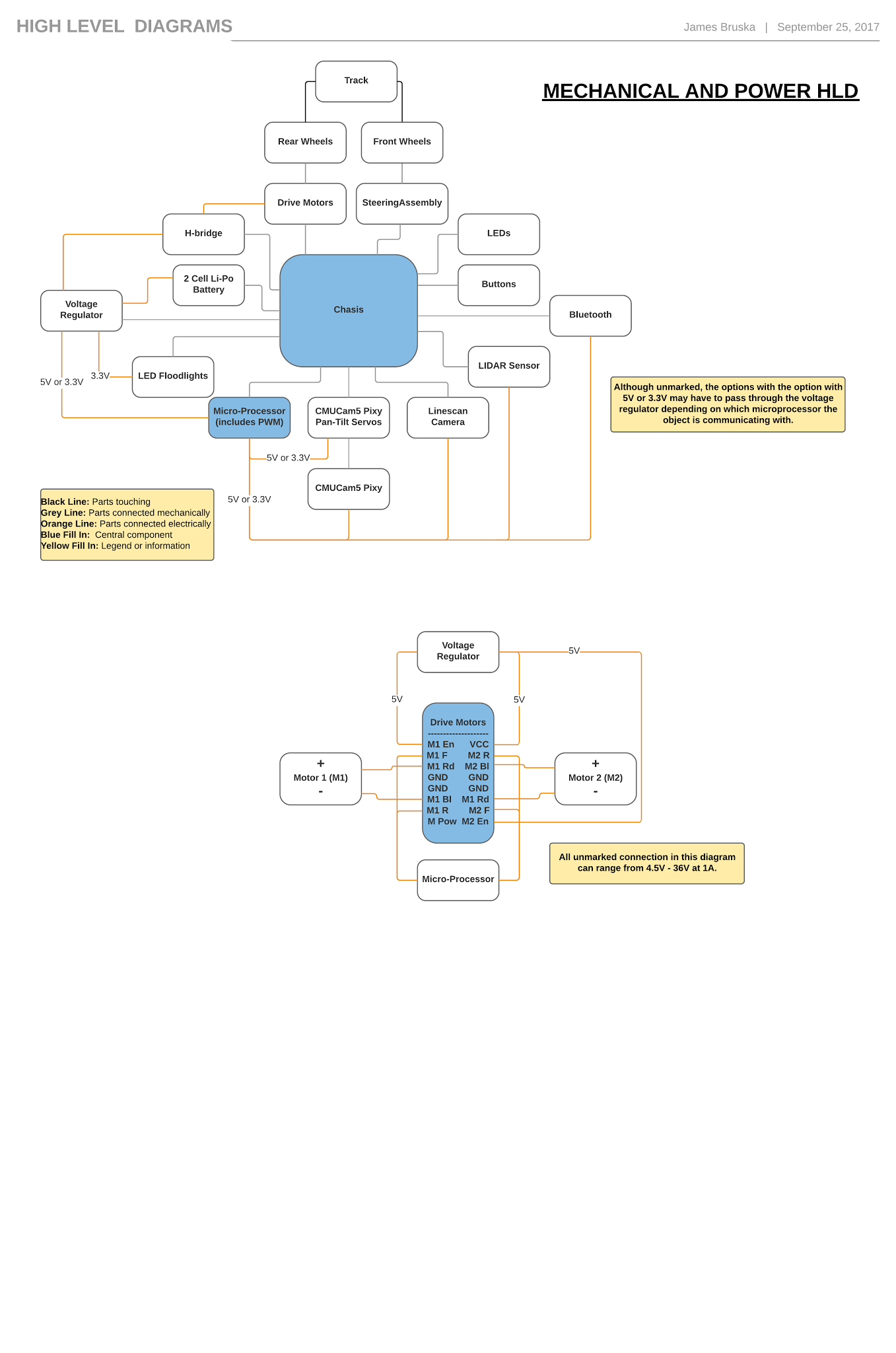
## Software High Level Design

****

## Graphical User Interface High Level Design

****

## Hardware High Level Diagram

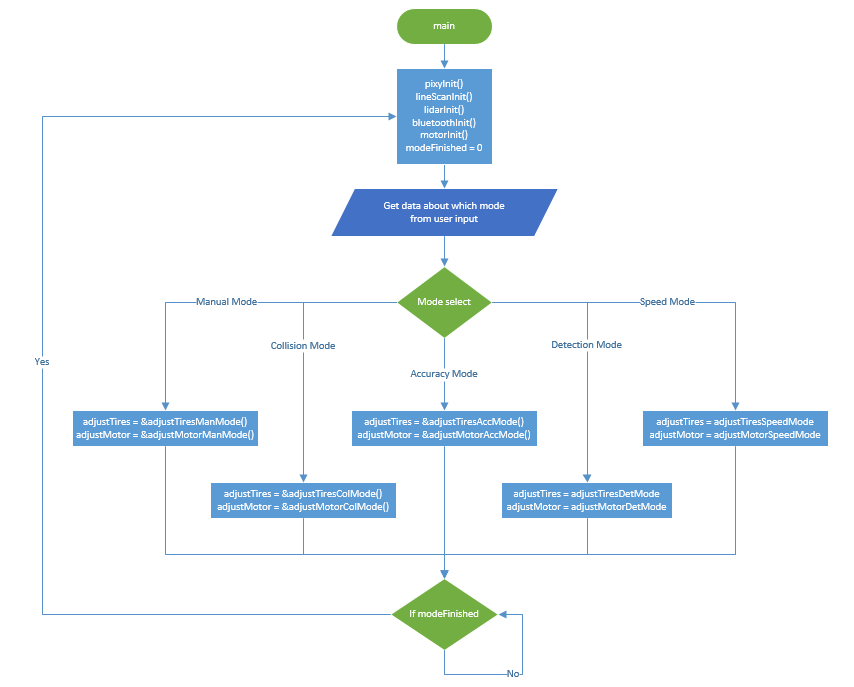
****

# 

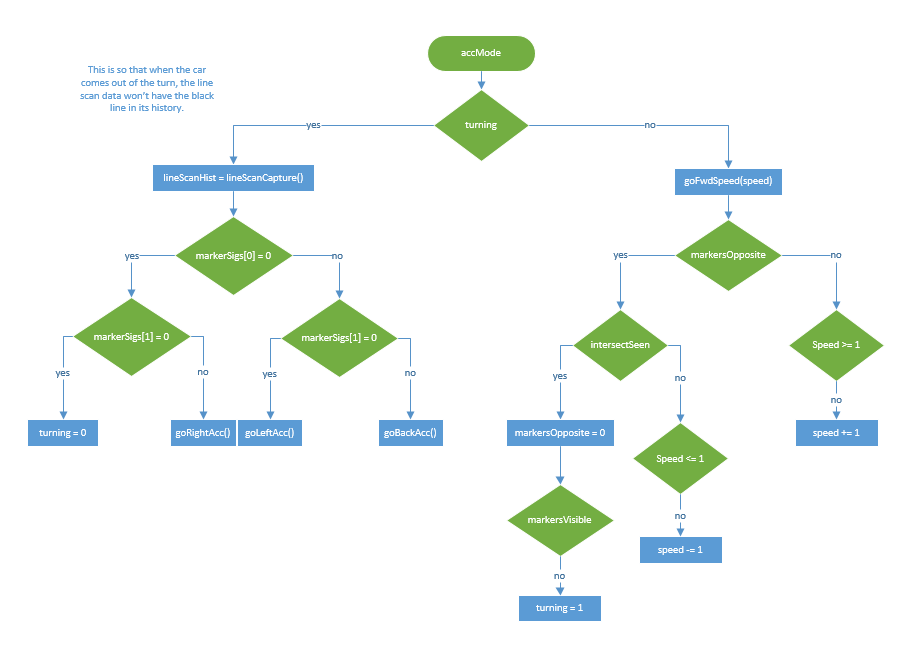
# Software Detailed Design

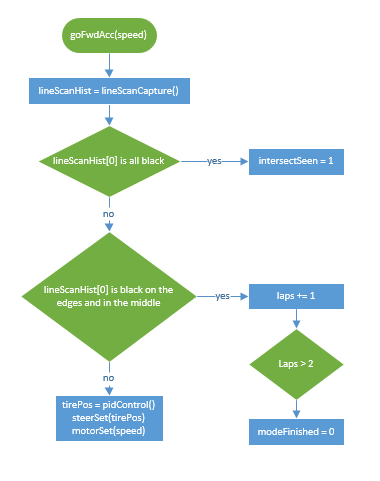
## Component Diagrams

### Main

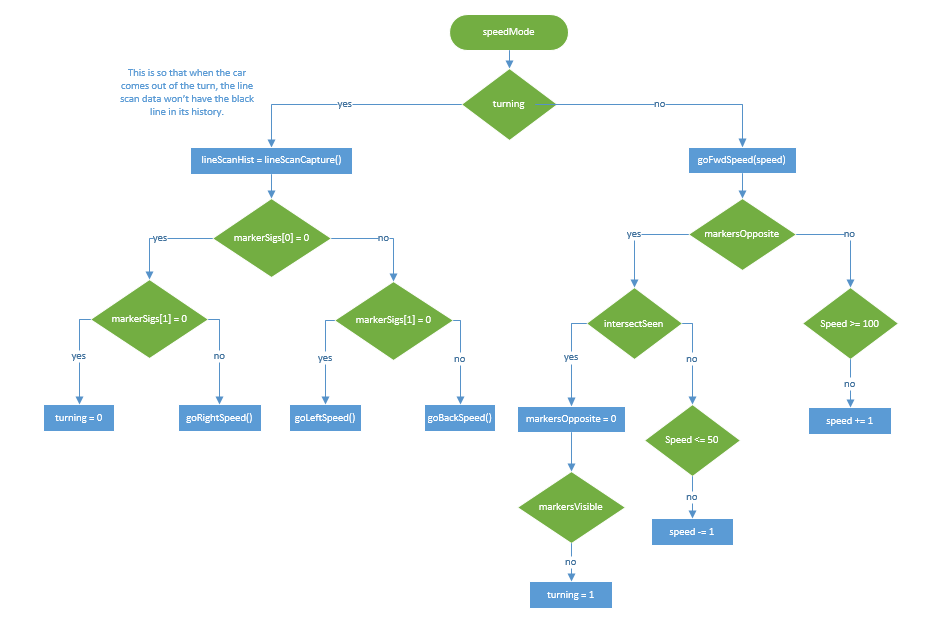


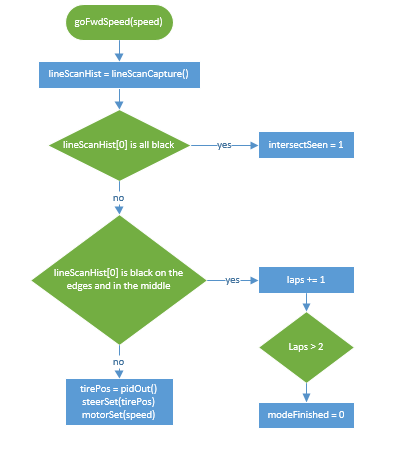
### Accuracy Mode



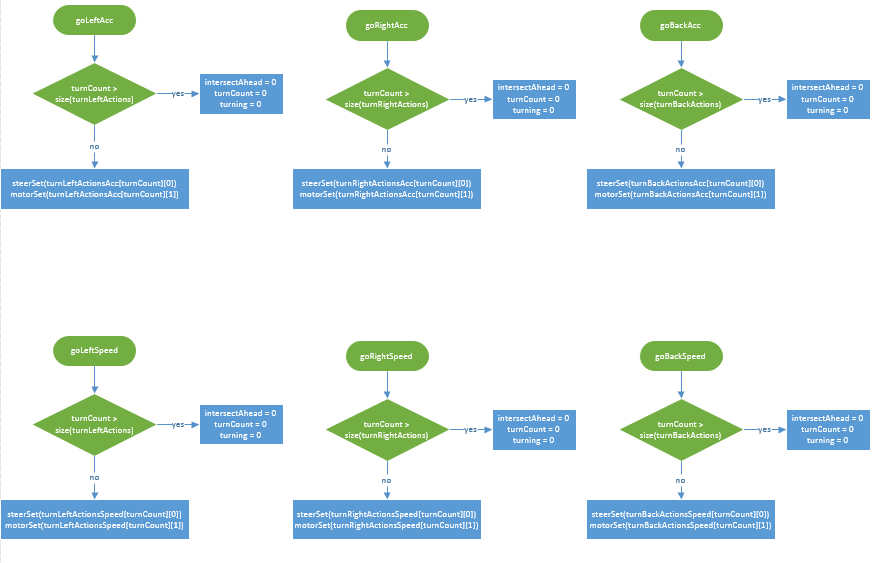


### Speed Mode





### Turning Mechanisms



## Zybo Components

### Visual Sensor Driver

#### Functional Purpose

The visual sensor driver shall be used to gather information from the visual sensor about what directional markers are in front of the car. It will then make a decision about how to act at the next intersection.

#### Functional Specification

The camera will be connected to the microcontroller over Arduino

ICSP SPI. This version of SPI does not need a slave select. The

Pixy camera will need the microcontroller’s MISO, MOSI, SCK, and

ground. The data is sent every 20ms. The blocks from the visual sensor are 16-bit and little endian. The visual sensor also outputs an extra sync word (0xaa55) at the start of a the next object block.

|  |  |  |
| --- | --- | --- |
| **Bytes** | **16-bit word** | **Description** |
| 0, 1 | y | Sync: 0xAA55=normal object, 0xAA56=color code object |
| 2, 3 | y | Checksum (sum of all 16-bit words 2-6, that is, bytes 4-13) |
| 4, 5 | y | Signature number |
| 6, 7 | y | X center of object |
| 8, 9 | y | Y center of object |
| 10, 11 | y | Width of object |
| 12, 13 | y | Height of object |

#### 

#### Component Function Designs

|  |  |
| --- | --- |
| **Function** | **Function Description** |
| pixyInit() | 1. Initialize SPI components 2. genTimer(20ms, Timer, pixyInterrupt) |
| pixyInterrupt() | 1. markerInformation = getVisualData() 2. If markerInformation is not an empty array then set markerVisible to 1 else set markerVisible to 0 3. If there are two 2 visual markers and they are not on the same side of the visual sensor input then markersOpposite to 1 |
| getVisualData() | 1. Call SPI protocol function defined for the board to gather the data from the visual sensor 2. Return the information |

#### Component Variable Designs

|  |  |
| --- | --- |
| **Variable** | **Variable Description** |
| SPI Variables | The variables needed for SPI communication with the board. These are defined in the board examples |
| Timer | The timer variable for how often the master should poll for new visual sensor data |
| markersOpposite | This is a flag describing if the last seen markers were on opposite sides of the screen |
| markerVisible | This describes if the marker is still visible by the visual sensor. If it is still visible then it will not turn at intersection. |
| markerInformation | This is an array to store marker data when the visual sensor sees a marker |

#### Hardware Connections

In order for the camera to be able to operate, the

microcontroller will need to supply 5v to the camera.

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Pin** |
| MISO | Input | JF04 (MIO-12) |
| MOSI | Output | JF03 (MIO-11) |
| SCK | Output | JF02 (MIO-10) |
| SS | N.A. | N.A. |

### 

### 

### 

### 

### Pan-Tilt Servo Driver [Design Phase 2]

#### Functional Purpose

Generic driver to adjusts the bearing of the Pixy camera relative to the front of the vehicle. The driver allows the microcontroller to command the servo to changing its viewing direction.

#### Functional Specification

The pan/tilt mechanism gets plugged directly into the Pixy Camera’s six servo pins and thus will be controlled using the same pins as the visual sensor.

#### Component Function Designs

#### Component Variable Designs

#### Internal Function Designs

#### Internal Variable Designs

#### Hardware Connections

### Linescan Camera Driver

#### Functional Purpose

The purpose of the line scan camera driver is to provide the generic driver for which the line scan camera will capture, process, and send information to both boards. The driver allows the line scan camera to create 1x128 pixel resolution image captures which will form the basis of the steering mechanism logic by enabling the car to view where its position is in relation to the track.

#### Functional Specification

The camera has two input signals and one output signal, as well as a supplied voltage to power the board of the camera itself. These signals, and how they interact with each board, are outlined below:

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Description** |
| SI | 1 input pulse high at beginning of capture cycle, low at any other time. PWM Frequency will be 50 Hz. | Each pulse will signal to the line scan camera to begin capturing. |
| CLK | Input Range: 5 kHz to 8 MHz  Selected: 6.45 kHz | Created by the microcontroller and output to the camera as a PWM. Allows the camera to capture for 129 cycles, then shuts off after the 130th. This allows the pixels to charge before the next capture, during which the clock will be turned on again, repeating this cycle for the duration of the mode. |
| Analog Out (AO) | Output Range: 0 to 3.3V | Analog representation of the 128 pixels captured by the camera. |

The two input signals, SI and CLK are purely for timing purposes during the capture process. CLK is the clock signal being sent from the microcontroller board, and SI is the enable to start image capture.

#### Component Function Designs

|  |  |
| --- | --- |
| **Function** | **Code Description** |
| lineScanInit | 1. Set up PWM with dutyCycle and interval 2. Set adjustTires = doNothing |
| lineScanCapture | 1. Send SI signal high to camera 2. Capture AO signal for 129 clock cycles 3. Use XADC to convert data to integer value AODig 4. Wait 1 clock cycle for pixels to charge 5. AOHist = lineScanGenHist when SI goes high 2nd time 6. Repeat |
| lineScanGenHist | 1. Enter AODig variable into integer array 2. Repeat 3. When AODig has been captured three times, AOReady is true |

There will be three functions necessary for the use of the line scan camera in conjunction with the microcontrollers.

The lineScanInit function will initialize all of the necessary variables to enable the microcontroller to use the line scan camera. These include the interval and duty cycle for the PWM used by the SI and CLK variables in the other functions listed here, as well as setting the universal adjustTires function to the doNothing state.

The lineScanCapture function will take in the board clock as an input variable. Using this clock as a output signal CLK, we will send a signal SI to start the data. The timing of the clock and this signal are outlined in the hardware design above. Once the SI signal is sent, the AO input signal is immediately captured for 129 clock cycles. Both the FRDM board and Zybo board uses an on-board XADC conversion to create a 32-bit unsigned integer array from the capture (AODig). The AOHist variable is returned from this function after the lineScanGenHist function creates a history of captures.

The lineScanGenHist function will take in the variable AODig in order to create a history of line scan captures for the board to use in steering logic. This function will be called by lineScanCapture. On each successive rising edge of SI, the AODig variable will be entered into the function for its center-point to be measured. The current center-point and two previous center-points will be used as an integer array to use in the steering mechanism logic. The AOReady variable will be used to signal that three captures have been made available and so that data is usable.

#### Component Variable Designs

|  |  |
| --- | --- |
| **Variable** | **Variable Description** |
| Clk | The clock signal from the board |
| dutyCycle | The duty cycle for the PWM used by both the SI and CLK signals |
| interval | The frequency at which the PWM runs at for the SI and CLK signals |
| SILocal | Local variable to be sent out when SI is needed to go high |
| AODig | The digitally converted version of the analog out signal of the line scan camera |
| AOHist | A history of the current and past two captures of the analog out signal of the line scan camera |
| AOReady | Goes high when an analog out history has been captured for three successive images in full |

#### Hardware Connections

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Zybo Pins** |
| SI | Input | JA1 (N15) |
| CLK | Input | JA2 (L14) |
| Analog Out (AO) | Output | JA3 (K16) |
| Supply Voltage | Input | JA6 |

### Motor Driver

#### Functional Purpose

Generic driver to adjust the speed and direction of the vehicle's onboard motors. The driver allows the microcontroller to command the MDD10A.

#### Functional Specification

The H-bridge takes two inputs from the board. These inputs are outlined below:

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Description** |
| PWM1/PWM2 | 20 kHz | PWM sent to the H-Bridge to control the motors |
| DIR1/DIR2 | 0 or 1 | Bit which is set to 0 or 1 depending on the desired direction of the car (forward or reverse) |

#### Component Function Designs

|  |  |
| --- | --- |
| **Function** | **Code Description** |
| motorInit() | 1. Set up PWM with dutyCycle and interval 2. Set adjustMotor = doNothing |
| motorSet() | 1. Take input of desired motor speed 2. Calculate PWM duty cycle 3. Adjust PWM with new duty cycle |

#### Component Variable Designs

|  |  |
| --- | --- |
| **Variable** | **Variable Description** |
| dutyCycle | The dutyCycle of the PWM for the H-bridge |
| interval | The frequency of the PWM for the H-bridge |
| motorSpeed | The input motorSpeed to be translated into a usable PWM duty cycle. |

#### Hardware Connections

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Zybo Pins** |
| PWM1/PWM2 | Input | JB1 (T20) |
| DIR1/DIR2 | Input | JB2 (U20) |

### Steering Servo Driver

#### Functional Purpose

Generic driver to adjust forward steering. Allows the controller to command the Futaba s3010 servo connected to chassis.

#### Functional Specification

The microcontroller will send a PWM signal to the servo based on the input from the line scan camera, as outlined below. The steering control method implemented will be a PID (Proportional-Integral-Derivative) controller which will measure the current error from the center point and weight its corrective steering angle based on a history of error from the center point (see section 3.3.1 for details).

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Description** |
| steerPWM | PWM Frequency: 50 Hz | For this servo, a PWM frequency is generated at 50 Hz, with the duty cycle being modified to direct the tires in the desired direction |

#### Component Function Designs

|  |  |
| --- | --- |
| **Function** | **Code Description** |
| steerInit() | 1. Set up PWM with dutyCycle and interval |
| steerSet() | 1. Take input of desired steering angle 2. Calculate PWM duty cycle 3. Adjust PWM with new duty cycle |

#### Component Variable Designs

|  |  |
| --- | --- |
| **Variable** | **Variable Description** |
| dutyCycle | The duty cycle of the PWM used by the servo |
| interval | The frequency of the PWM used by the servo, set at 50 Hz during normal operation |
| tirePos | The input tire position to be translated into a usable PWM duty cycle |

#### Hardware Connections

|  |  |  |
| --- | --- | --- |
| Name | Type | Zybo Pins |
| steerPWM | Input | JB3 (V20) |

### LIDAR Driver [Design Phase 2]

#### Functional Purpose

This is a generic driver for communication with the LIDAR sensor. The driver analyses the sensor information to be used by the microcontroller.

#### Functional Specification

#### Component Function Designs

#### Component Variable Designs

#### Internal Function Designs

#### Internal Variable Designs

#### Hardware Connections

### Bluetooth Driver [Design Phase 2]

#### Functional Purpose

This is a generic driver for bluetooth communication. The driver allows the microcontroller to utilize a bluetooth interface and communicate with other bluetooth devices.

#### Functional Specification

#### Component Function Designs

#### Component Variable Designs

#### Internal Function Designs

#### Internal Variable Designs

#### Hardware Connections

## FRDM Components

### Visual Sensor Driver

#### Functional Purpose

The visual sensor driver shall be used to gather information from the visual sensor about what directional markers are in front of the car. It will then make a decision about how to act at the next intersection.

#### Functional Specification

The camera will be connected to the microcontroller over Arduino

ICSP SPI. This version of SPI does not need a slave select. The

Pixy camera will need the microcontroller’s MISO, MOSI, SCK, and

ground. The data is sent every 20ms. The blocks from the visual sensor are 16-bit and little endian. The visual sensor also outputs an extra sync word (0xaa55) at the start of a the next object block.

|  |  |  |
| --- | --- | --- |
| **Bytes** | **16-bit word** | **Description** |
| 0, 1 | y | Sync: 0xAA55=normal object, 0xAA56=color code object |
| 2, 3 | y | Checksum (sum of all 16-bit words 2-6, that is, bytes 4-13) |
| 4, 5 | y | Signature number |
| 6, 7 | y | X center of object |
| 8, 9 | y | Y center of object |
| 10, 11 | y | Width of object |
| 12, 13 | y | Height of object |

#### 

#### Component Function Designs

|  |  |
| --- | --- |
| **Function** | **Function Description** |
| pixyInit() | 1. Initialize SPI components 2. genTimer(20ms, Timer, pixyInterrupt) |
| pixyInterrupt() | 1. markerInformation = getVisualData() 2. If markerInformation is not an empty array then set markerVisible to 1 else set markerVisible to 0 3. If there are 2 visual markers and they are not on the same side of the visual sensor input then markersOpposite to 1 else set markersOpposite to 0 |
| getVisualData() | 1. Call SPI protocol function defined for the board to gather the data from the visual sensor 2. Return the information |

#### Component Variable Designs

|  |  |
| --- | --- |
| **Variable** | **Variable Description** |
| SPI Variables | The variables needed for SPI communication with the board. These are defined in the board examples |
| Timer | The timer variable for how often the master should poll for new visual sensor data |
| markersOpposite | This is a flag describing if the last seen markers were on opposite sides of the screen |
| markerVisible | This describes if the marker is still visible by the visual sensor. If it is still visible then it will not turn at intersection. |
| markerInformation | This is an array to store marker data when the visual sensor sees a marker |

#### Hardware Connections

In order for the camera to be able to operate, the

microcontroller will need to supply 5v to the camera.

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Pin** |
| MISO | Input | JF04 (MIO-12) |
| MOSI | Output | JF03 (MIO-11) |
| SCK | Output | JF02 (MIO-10) |
| SS | N.A. | N.A. |

### 

### Pan-Tilt Servo Driver [Design Phase 2]

#### Functional Purpose

Generic driver to adjusts the bearing of the Pixy camera relative to the front of the vehicle. The driver allows the microcontroller to command the servo to changing its viewing direction.

#### Functional Specification

The pan/tilt mechanism gets plugged directly into the Pixy Camera’s six servo pins and thus will be controlled using the same pins as the visual sensor.

#### Component Function Designs

#### Component Variable Designs

#### Internal Function Designs

#### Internal Variable Designs

#### Hardware Connections

### Line Scan Camera Driver

#### Functional Purpose

The purpose of the line scan camera driver is to provide the generic driver for which the line scan camera will capture, process, and send information to both boards. The driver allows the line scan camera to create 1x128 pixel resolution image captures which will form the basis of the steering mechanism logic by enabling the car to view where its position is in relation to the track.

#### Functional Specification

The camera has two input signals and one output signal, as well as a supplied voltage to power the board of the camera itself. These signals, and how they interact with each board, are outlined below:

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Description** |
| SI | 1 input pulse high at beginning of capture cycle, low at any other time. PWM Frequency will be 50 Hz. | Each pulse will signal to the line scan camera to begin capturing. |
| CLK | Input Range: 5 kHz to 8 MHz  Selected: 6.45 kHz | Created by the microcontroller and output to the camera as a PWM. Allows the camera to capture for 129 cycles, then shuts off after the 130th. This allows the pixels to charge before the next capture, during which the clock will be turned on again, repeating this cycle for the duration of the mode. |
| Analog Out (AO) | Output Range: 0 to 3.3V | Analog representation of the 128 pixels captured by the camera. |

The two input signals, SI and CLK are purely for timing purposes during the capture process. CLK is the clock signal being sent from the microcontroller board, and SI is the enable to start image capture.

#### Component Function Designs

|  |  |
| --- | --- |
| **Function** | **Code Description** |
| lineScanInit | 1. Set up PWM with dutyCycle and interval 2. Set adjustTires = doNothing |
| lineScanCapture | 1. Send SI signal high to camera 2. Capture AO signal for 129 clock cycles 3. Use XADC to convert data to integer value AODig 4. Wait 1 clock cycle for pixels to charge 5. AOHist = lineScanGenHist when SI goes high 2nd time 6. Repeat |
| lineScanGenHist | 1. Enter AODig variable into integer array 2. Repeat 3. When AODig has been captured three times, AOReady is true |

There will be three functions necessary for the use of the line scan camera in conjunction with the microcontrollers.

The lineScanInit function will initialize all of the necessary variables to enable the microcontroller to use the line scan camera. These include the interval and duty cycle for the PWM used by the SI and CLK signals in the other functions listed here, as well as setting the universal adjustTires function to the doNothing state.

The lineScanCapture function will take in the board clock Clk as an input variable. Using this clock as a output signal CLK, we will send a signal SI to start the data. The timing of the clock and this signal are outlined in the hardware design above. Once the SI signal is sent, the AO input signal is immediately captured for 129 clock cycles. Both the FRDM board and Zybo board uses an on-board XADC conversion to create a 32-bit unsigned integer array from the capture (AODig). The AOHist variable is returned from this function after the lineScanGenHist function creates a history of captures.

The lineScanGenHist function will take in the variable AODig in order to create a history of line scan captures for the board to use in steering logic. This function will be called by lineScanCapture. On each successive rising edge of SI, the AODig variable will be entered into the function for its center-point to be measured. The current center-point and two previous center-points will be used as an integer array to use in the steering mechanism logic. The AOReady variable will be used to signal that three captures have been made available and so that data is usable.

#### Component Variable Designs

|  |  |
| --- | --- |
| **Variable** | **Variable Description** |
| Clk | The clock signal from the board |
| dutyCycle | The duty cycle for the PWM used by both the SI and CLK signals |
| interval | The frequency at which the PWM runs at for the SI and CLK signals |
| SILocal | Local variable to be sent out when SI is needed to go high |
| AODig | The digitally converted version of the analog out signal of the line scan camera |
| AOHist | A history of the current and past two captures of the analog out signal of the line scan camera |
| AOReady | Goes high when an analog out history has been captured for three successive images in full |

#### Hardware Connections

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **FRDM Pins** |
| SI | Input | PTC10 (FTM3\_CH6) |
| CLK | Input | PTC11 (FTM3\_CH7) |
| Analog Out (AO) | Output | PTB2 (ADC0\_SE12) |
| Supply Voltage | Input | VOUT33 |

### Motor Driver

#### Functional Purpose

Generic driver to adjust the speed and direction of the vehicle's onboard motors. The driver allows the microcontroller to command the MDD10A.

#### Functional Specification

The H-bridge takes two inputs from the board. These inputs are outlined below:

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Description** |
| PWM1/PWM2 | 20 kHz | PWM sent to the H-Bridge to control the motors |
| DIR1/DIR2 | 0 or 1 | Bit which is set to 0 or 1 depending on the desired direction of the car (forward or reverse) |

#### Component Function Designs

|  |  |
| --- | --- |
| **Function** | **Code Description** |
| motorInit() | 1. Set up PWM with dutyCycle and interval 2. Set adjustMotor = doNothing |
| motorSet() | 1. Take input of desired motor speed 2. Calculate PWM duty cycle 3. Adjust PWM with new duty cycle |

#### Component Variable Designs

|  |  |
| --- | --- |
| **Variable** | **Variable Description** |
| dutyCycle | The dutyCycle of the PWM for the H-bridge |
| interval | The frequency of the PWM for the H-bridge |
| motorSpeed | The input motorSpeed to be translated into a usable PWM duty cycle. |

#### Hardware Connections

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **FRDM Pins** |
| PWM1/PWM2 | Input | PTD0 (FTM3\_CH0) |
| DIR1/DIR2 | Input | PTD3 |

### Steering Servo Driver

#### Functional Purpose

Generic driver to adjust forward steering. Allows the controller to command the Futaba s3010 servo connected to chassis.

#### Functional Specification

The microcontroller will send a PWM signal to the servo based on the input from the line scan camera, as outlined below. The steering control method implemented will be a PID (Proportional-Integral-Derivative) controller which will measure the current error from the center point and weight its corrective steering angle based on a history of error from the center point (see section 3.3.1 for details).

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Description** |
| steerPWM | PWM Frequency: 50 Hz | For this servo, a PWM frequency is generated at 50 Hz, with the duty cycle being modified to direct the tires in the desired direction |

#### Component Function Designs

|  |  |
| --- | --- |
| **Function** | **Code Description** |
| steerInit() | 1. Set up PWM with dutyCycle and interval |
| steerSet() | 1. Take input of desired steering angle 2. Calculate PWM duty cycle 3. Adjust PWM with new duty cycle |

#### Component Variable Designs

|  |  |
| --- | --- |
| **Variable** | **Variable Description** |
| dutyCycle | The duty cycle of the PWM used by the servo |
| interval | The frequency of the PWM used by the servo, set at 50 Hz during normal operation |
| tirePos | The input tire position to be translated into a usable PWM duty cycle |

#### Hardware Connections

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **FRDM Pin** |
| steerPWM | Input | Connected to PWM pin PTA1 (FTM0\_CH6) |

### LIDAR Driver [Design Phase 2]

#### Functional Purpose

This is a generic driver for communication with the LIDAR sensor. The driver analyses the sensor information to be used by the microcontroller.

#### Functional Specification

#### Component Function Designs

#### Component Variable Designs

#### Internal Function Designs

#### Internal Variable Designs

#### Hardware Connections

### Bluetooth Driver [Design Phase 2]

#### Functional Purpose

This is a generic driver for bluetooth communication. The driver allows the microcontroller to utilize a bluetooth interface and communicate with other bluetooth devices.

#### Functional Specification

#### Component Function Designs

#### Component Variable Designs

#### Internal Function Designs

#### Internal Variable Designs

#### Hardware Connections

## Generic Components

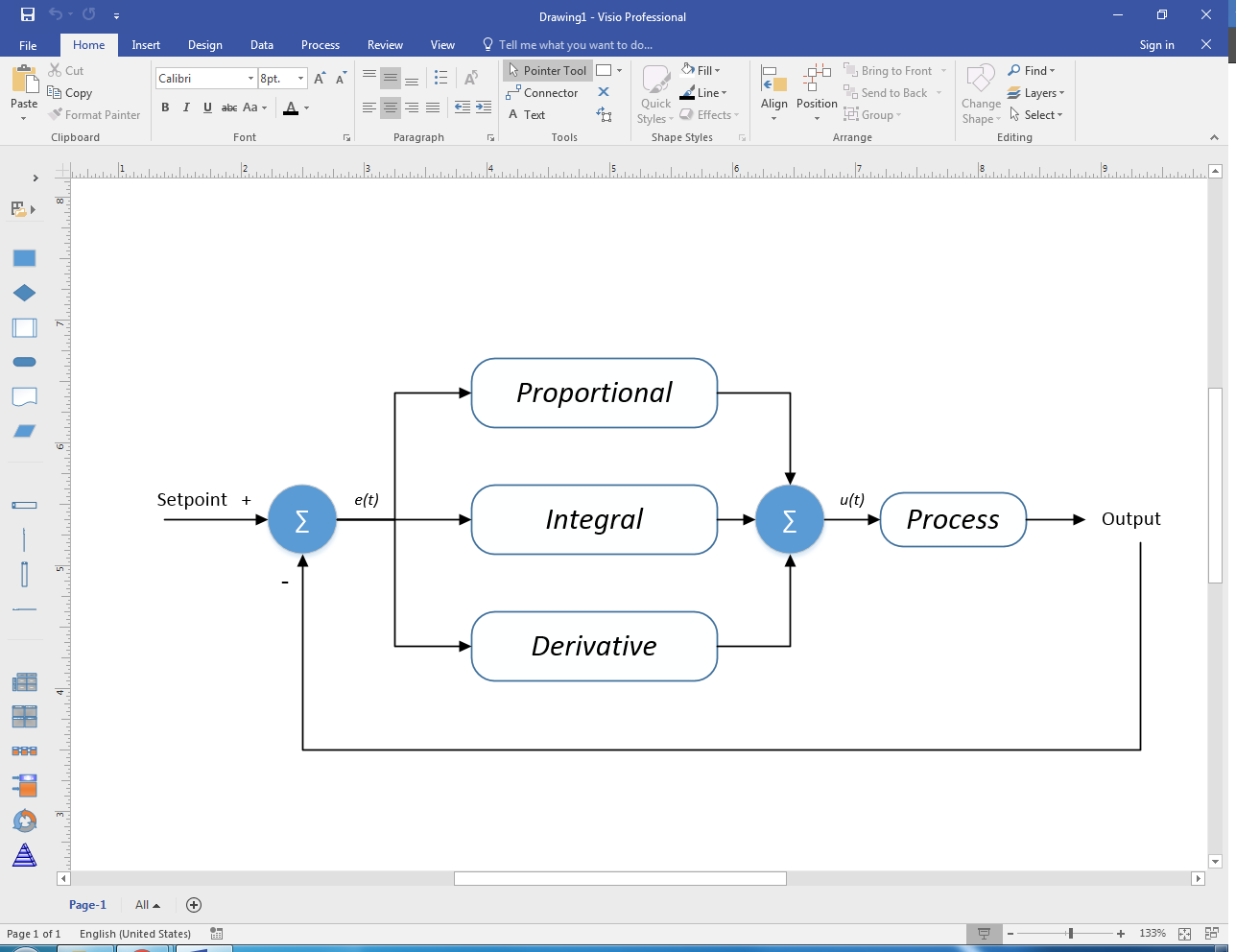
### Line Following Mechanism

#### Functional Purpose

The purpose of the line following mechanism of the car is to allow the car to autonomously follow the track with no forms of input other than the information gathered by the car itself about the track. The mechanism allows the wheels to be steered in the direction to best place the track directly in the center of the car as it is driven over. The mechanism is controlled by a PID controller. In its generalized form, a PID controller acts as a control system for an ongoing feedback loop. The three portions of a PID controller (P, I, and D, respectively) each serve their own individual purpose in the control system that balance out the faults of the other portions.

#### Functional Specification

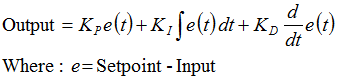
The diagram below shows the logic which forms the basic structure of the PID controller:



The table below explains this system in more detail:

|  |  |  |
| --- | --- | --- |
| Type | Purpose | Faults |
| Proportional | Acts as the main corrective action as the control system. Gives a response which is proportional to the degree of error from the set point (desired value). | Tends towards large amounts of overshoot. Needs input error for a response, therefore a certain degree of error will always exist when using this alone as a control method. |
| Integral | Acts as a historical record of previous error. This serves to eliminate the need for a constant error value in a proportional-only system by observing the previous values and adjusting the correction based on those values. | Slow to react when used as a standalone unless starting from a large amount of error. Corrective action can be very severe when the error history is still positive in nature. |
| Derivative | Acts as a damping force on the corrective action by observing the rate of change of the error value. Where a PI-only system would likely overshoot its set point multiple times while trying to issue a corrective action, the PID system (when tuned properly) would likely only overshoot once or not at all. | The damping effect of the derivative must be tuned in order for the system to function as intended. A large amount of damping may end in more overshoot than previously allowed, or even a completely unstable system where the error grows larger over time. |

The following equation must be utilized by the autonomous steering function of the microcontroller in order to act as a proper PID controller:



The output in this case is the servo position of the steering mechanism. The terms *KP, KI,* and *KD* are the constants of the proportional, integral, and derivative terms, respectively. With this equation the basis of the autonomous steering function is formed.

#### Component Function Designs

|  |  |
| --- | --- |
| **Function** | **Code Description** |
| PIDOut() | 1. tirePos = PIDControl equation 2. Return tirePos |

#### Component Variable Designs

|  |  |
| --- | --- |
| **Variable** | **Variable Description** |
| tirePos | The output of the PID control equation determining the tire position |
| pGain | The gain of the proportional term of the PID equation determining the tire position |
| iGain1 | The gain of the first integral term of the PID equation determining the tire position |
| iGain2 | The gain of the second integral term of the PID equation determining the tire position |
| dGain | The gain of the derivative term of the PID equation determining the tire position |
| tirePosPrev | The previous output of the tire position. Necessary for the derivative term of the PID equation to determine tire position |

### Turning Mechanism

#### Functional Purpose

The purpose of the line turning mechanism of the car is to allow the car to turn once an intersection has been reached. The mechanism allows the wheels to be steered in the direction to best place the track directly in the center of the car as it is driven around the turn.

#### Functional Specification

These are the simplistic functions that will take a value from the array for a current iteration and then update the tire positions and speed based on those values. The values in the array will be determined by trial and error.

#### Component Function Designs

|  |  |
| --- | --- |
| **Function** | **Function Description** |
| goLeftAcc | 1. If turnCount > size of turnLeftActionsAcc    1. Then intersectAhead = 0, turnCount = 0, and turning = 0    2. Else steerSet(turnLeftActionsAcc[turnCount][0]) and  motorSet(turnLeftActionsAcc[turnCount][1]) |
| goRightAcc | 1. If turnCount > size of turnRightActionsAcc    1. Then intersectAhead = 0, turnCount = 0, and turning = 0    2. steerSet(turnRightActionsAcc[turnCount][0]) and motorSet(turnRightActionsAcc[turnCount][1]) |
| goBackAcc | 1. If turnCount > size of turnBackActionsAcc    1. Then intersectAhead = 0, turnCount = 0, and turning = 0    2. Else steerSet(turnBackActionsAcc[turnCount][0]) and motorSet(turnBackActionsAcc[turnCount][1]) |
| goLeftSpeed | 1. If turnCount > size of turnLeftActionsSpeed    1. Then intersectAhead = 0, turnCount = 0, and turning = 0    2. Else steerSet(turnLeftActionsSpeed[turnCount][0]) and motorSet(turnLeftActionsSpeed[turnCount][1]) |
| goRightSpeed | 1. If turnCount > size of turnRightActionsSpeed    1. Then intersectAhead = 0, turnCount = 0, and turning = 0    2. Else steerSet(turnRightActionsSpeed[turnCount][0]) and motorSet(turnRightActionsSpeed[turnCount][1]) |
| goBackSpeed | 1. If turnCount > size of turnBackActionsSpeed    1. Then intersectAhead = 0, turnCount = 0, and turning = 0    2. Else steerSet(turnBackActionsSpeed[turnCount][0]) and motorSet(turnBackActionsSpeed[turnCount][1]) |

#### Component Variable Designs

|  |  |
| --- | --- |
| **Variable** | **Variable Description** |
| turnLeftActionsAcc | A 2D array that contains the speed and tire direction for a left hand turn in Accuracy Mode |
| turnRightActionsAcc | A 2D array that contains the speed and tire direction for a right hand turn in Accuracy Mode |
| turnBackActionsAcc | A 2D array that contains the speed and tire direction to go turn in Accuracy Mode |
| turnLeftActionsSpeed | A 2D array that contains the speed and tire direction for a left hand turn in Speed Mode |
| turnRightActionsSpeed | A 2D array that contains the speed and tire direction for a right hand turn in Speed Mode |
| turnBackActionsSpeed | A 2D array that contains the speed and tire direction to go turn in Speed Mode |
| turnCount | Describes what iteration of the turn array the algorithm is on |
| intersectAhead | Flag marking if there is an intersection ahead |
| turning | Flag marking if I am in a turn algorithm |

### Accuracy Mode Logic

#### Functional Purpose

The purpose of the logic in this operational mode is to allow the car to complete two laps around the track in a manner that is accurate to the track’s path as possible. This means that at no point during those two laps will there be more than three wheels of the car on one side of the track. When presented with the colored markers, the car will perform the action which is indicated by the colors on the markers (at the next intersection, either continue forward, turn around, turn left, or turn right).

#### Functional Specification

In accuracy mode, per KVM Requirements Specifications 3.3.6, the car shall perform in the following manner:

|  |  |
| --- | --- |
| **Requirements Specification Number** | **Description** |
| **3.3.6.1** | When in this mode, no external interaction will be needed to perform any of the other requirements. |
| **3.3.6.2** | This operational mode requires that the car is placed on the starting line of the track. |
| **3.3.6.3** | Upon the initialization of accuracy mode, the car will begin to move forward around the track. |
| **3.3.6.4** | The car shall follow the instructions of the directional markers until it has performed two complete laps. |
| **3.3.6.5** | Upon completion of the second lap, the car shall stop within one meter of the stop line. |
| **3.3.6.6** | The car shall not be required to move at any specific speed, so long it successfully performs the previously mentioned tasks for this mode. |

#### 

#### 

#### 

#### Component Function Designs

|  |  |
| --- | --- |
| **Function** | **Function Description** |
| accMode | 1. If turning    1. Then lineScanHist = linescanCapture() and if markerSigs[0] = 0       1. Then if markerSigs[1] = 0          1. Then turning = 1          2. Else goRightAcc()       2. Else if markerSigs[0] = 1          1. Then goLeftAcc()          2. Else goBackAcc()    2. Else goFwdAcc(speed) and If markersOpposite       1. Then if intersectSeen          1. Then markersOpposite = 0 and if !markersVisible             1. Then turning = 1          2. Else if !(speed <= 1)             1. Then speed -= 1       2. Else if !(speed >= 1)          1. Then speed += 1 |
| goFwdAcc(speed) | 1. lineScanHist = linescanCapture() 2. If lineScanHist[0] is all black    1. Then intersectSeen = 1    2. Else if lineScanHist is black on edges and in middle       1. Laps += 1 and if laps > 2          1. Then modeFinished = 0       2. Else tirePos = pidOut(), steerSet(tirePos), and motorSet(speed) |

#### Component Variable Designs

|  |  |
| --- | --- |
| **Variable** | **Variable Description** |
| turning | A flag that indicates if the car is turning |
| speed | A variable that represents the speed the car should move when going forward |
| markerSigs | The signatures of the most recent markers seen. Used for deciding which way to turn |
| intersectSeen | A flag for if the line scan has seen the intersection |
| markersVisible | A flag for if the markers are still visible |
| tirePos | Output signal for the servo position controlling the steering wheels of the car |
| lineScanHist | Array of previous lineScan data |

### Speed Mode Logic

#### Functional Purpose

The purpose of the logic in this operational mode is to allow the car to complete two laps around the track as quickly as possible. When presented with the colored markers, the car will perform the action which is indicated by the colors on the markers (at the next intersection, either continue forward, turn around, turn left, or turn right).

#### Functional Specification

In speed mode, per KVM Requirements Specifications 3.3.7, the car shall perform in the following manner:

|  |  |
| --- | --- |
| **Function** | **Function Description** |
| speedMode | 1. If turning    1. Then lineScanHist = linescanCapture() and if markerSigs[0] = 0       1. Then if markerSigs[1] = 0          1. Then turning = 1          2. Else goRightSpeed()       2. Else if markerSigs[0] = 1          1. Then goLeftSpeed()          2. Else goBackSpeed()    2. Else goFwdAcc(speed) and If markersOpposite       1. Then if intersectSeen          1. Then markersOpposite = 0 and if !markersVisible             1. Then turning = 1          2. Else if !(speed <= 50)             1. Then speed -= 1       2. Else if !(speed >= 100)          1. Then speed += 1 |
| goFwdSpeed(speed) | 1. lineScanHist = linescanCapture() 2. If lineScanHist[0] is all black    1. Then intersectSeen = 1    2. Else if lineScanHist is black on edges and in middle       1. Laps += 1 and if laps > 2          1. Then modeFinished = 0       2. Else tirePos = pidOut(), steerSet(tirePos), and motorSet(speed) |

#### Component Variable Designs

|  |  |
| --- | --- |
| **Variable** | **Variable Description** |
| turning | A flag that indicates if the car is turning |
| speed | A variable that represents the speed the car should move when going forward |
| markerSigs | The signatures of the most recent markers seen. Used for deciding which way to turn |
| intersectSeen | A flag for if the line scan has seen the intersection |
| markersVisible | A flag for if the markers are still visible |
| tirePos | Output signal for the servo position controlling the steering wheels of the car |
| lineScanHist | Array of previous lineScan data |

### Discovery Mode Logic [Design Phase 2]

#### Functional Purpose

This mode allows the vehicle to start off track. The vehicle searches for the track and follows once the track is found.

#### Functional Specification

#### Component Function Designs

#### Component Variable Designs

#### Internal Function Designs

#### Internal Variable Designs

### Collision Mode Logic [Design Phase 2]

#### Functional Purpose

The vehicle follows the track, making efforts to avoid rear end collisions with forward cars.

#### Functional Specification

#### Component Function Designs

#### Component Variable Designs

#### Internal Function Designs

#### Internal Variable Designs

### Manual Mode Logic [Design Phase 2]

#### Functional Purpose

The vehicle is run from an external source.

#### Functional Specification

#### Component Function Designs

#### Component Variable Designs

#### Internal Function Designs

#### Internal Variable Designs

### Graphical User Interface

#### Functional Purpose

The GUI allows for car monitoring and control during manual mode.

#### Functional Specification

When the user runs the GUI software a menu of available actions will appear, including one checkbox to toggle the power labeled “Power On” if the vehicle is powered on and “Power Off” if the vehicle is powered off. The menu will also have a radio button group that the user can select from one of the following operational modes: “Neutral”, “Discovery Mode”, “Accuracy Mode”, “Speed Mode”, “Collision Avoidance Mode”, and “Manual Mode”. The GUI should change the operational mode of the vehicle within one second of that operational mode being selected. When the user clicks the Checkbox labeled “Power On” the car will turn off remotely within one second and the checkbox will be labeled “Power Off” and the checkbox and radio button group will be disabled until the vehicle is powered on manually.

The GUI will also have a slider that is horizontal and one that is vertical, next to each other. The slider that is horizontal will be used to control the direction of the vehicle in manual mode, and the slider that is vertical will be used to control the speed of the vehicle in manual mode. When the vehicle is not in manual operational mode, the sliders will reflect the current direction and speed of the vehicle in one second intervals.

The GUI will also show a speed versus time graph and direction versus time graph. The speed versus time graph will have an automatically scaling y-axis and will adjust to the current maximum speed, the direction versus time graph will just show the min to max range of direction values. The graphs will both update in one second intervals.

#### Interface Signals and variables specification

The GUI will use PySerial and communicate serially via the PC comm port using the Tx and Rx pins of the “Bluetooth USB Module Mini – WRL-09434” and communicate with a “RedBearLab BLE Nano v2 – nRF52832” connected to the microprocessor. The GUI will use Tkinter to display Tk GUI widgets. The GUI will use Matplotlib to display graphical information.

#### Data items needed

|  |  |  |
| --- | --- | --- |
| **Class** | **Function** | **Variables** |
| KVMGUI(tk.Tk) |  | container(tk.Frame)  frames(tk.Frame [])  carMode(integer)  powerState(boolean) |
|  | show\_frame(self,page\_name) | frame(tk.Frame) |
|  | serialInput(self) | inputInstruction(string)  direction(integer) directions(integer [])  speed(integer)  speeds(integer [])  maxSpeed(integer) |
|  | serialOutput(self, outputInstruction(string)) |  |
| ModeSelect(tk.Frame) |  | powerCheckButton( tk.Checkbutton)  inactiveRadioButton( tk.Radiobutton)  discoveryRadioButton( tk.Radiobutton)  accuracyRadioButton( tk.RadioButton)  collisionRadioButton( tk.RadioButton)  manualRadioButton( tk.RadioButton) |
|  | changeMode(self) |  |
| ManualControl(tk.Frame) |  | direction(tk.Scale)  speed(tk.Scale) |
|  | updateVehicleDirection(self) |  |
|  | updateVehicleSpeed(self) |  |
|  | updateDirectionScale(self,value) |  |
|  | updateSpeedScale(self,value) |  |
| SpeedGraph(tk.Frame) |  | canvas(  FigureCanvasTkAgg)  toolbar( NavigationTOllbar2TkAgg) |
|  | updateSpeedGraph(self, speeds(integer [])) |  |
| DirectionGraph(tk.Frame) |  | canvas(  FigureCanvasTkAgg)  toolbar( NavigationTOllbar2TkAgg) |
|  | updateDirectionGraph( directions(integer [])) |  |

#### 

#### 

#### 

#### Pseudo Code

|  |  |
| --- | --- |
| **Instruction type** | **Binary requirement** |
| Read speed | First bit low to indicate read, second and 3rd bit low to indicate speed, last 9 bits for data |
| Read direction | First bit low to indicate read, second bit low and 3rd bit high to indicate direction, last 9 bits for data |
| Read – change mode | First bit low to indicate read, second bit high and 3rd bit low to indicate change mode, 4th bit to change on or off, last 3 bits for mode |
| Write speed | First bit high to indicate write, second and 3rd bit low to indicate speed, last 9 bits for data |
| Write Direction | First bit high to indicate write, second bit low and 3rd bit high to indicate direction, last 9 bits for data |
| Write - change mode | First bit high to indicate write, second bit high and 3rd bit low to indicate change mode, 4th bit to change on or off, last 3 bits for mode |

|  |  |
| --- | --- |
| **Function** | **Pseudo Code** |
| showFrame(self, page\_name) | 1. Call tkraise() on the tk.Frame object in the frames dictionary with page\_name as the key |
| serialInput(self) | 1. Read inputInstruction from the serial comm port using PySerial 2. Check if the instruction is a speed or direction 3. If it is a speed instruction, then append the speed to speeds(integer []) and call updateSpeedGraph() with the new speeds array 4. Else if it is a direction instruction, then append the direction to directions(integer []) and call updateDirectionGraph() with the new directions array. |
| serialOutput(self, outputInstruction) | 1. Send outputInstruction to the serial comm port using PySerial |
| changeMode(self) | 1. check if the powerState is off, if so disable the power Checkbutton and the Radiobuttons in the ModeSelect frame 2. else if the powerState is on, check the value of carMode and if it is 0 then call serialOutput with an instruction to change to neutral mode and update the GUI to reflect this change of state 3. else if the powerState is on, check the value of carMode and if it is 1 then call serialOutput with an instruction to change to discovery mode and update the GUI to reflect this change of state 4. else if the powerState is on, check the value of carMode and if it is 2 then call serialOutput with an instruction to change to accuracy mode and update the GUI to reflect this change of state 5. else if the powerState is on, check the value of carMode and if it is 3 then call serialOutput with an instruction to change to collision avoidance mode and update the GUI to reflect this change of state 6. else if the powerState is on, check the value of carMode and if it is 4 then call serialOutput with an instruction to change to manual mode and update the GUI to reflect this change of state |
| updateVehicleDirection(self) | 1. write to serialOutput the current value of the direction(tk.Scale) widget |
| updateVehicleSpeed(self) | 1. write to serialOutput the current value of the speed(tk.scale) widget |
| updateDirectionScale(self,value) | 1. set the direction(tk.Scale) widget to “value” |
| updateSpeedScale(self,value) | 1. set the speed(tk.Scale) widget to “value” |
| updateSpeedGraph(self, speeds(integer [])) | 1. resize the y-axis so that the maxSpeed(integer) is 2. replot the scatter plot with the speeds array |
| updateDirectionGraph(self, directions(integer [])) | 1. replot the scatter plot with the directions array |

## 

# Hardware Detailed Design

## Zybo Micro-Controller

### Part Name

Zybo Zynq-7000 ARM/FPGA SoC Trainer Board

### Functional Specification

Act as the primary logic and control unit for the chassis, with multiple communication interfaces for interaction with sensors to receive data. These sensors include a Bluetooth module for manual input, a CMUcam5 Pixy Cam for object recognition, a TAOS line scanner for track recognition, and an LIDAR sensor for collision avoidance. The board also transmits signals to an H-Bridge motor driver which drives the wheels, a servo mechanism which controls the pan and tilt of the Pixy Cam, and another servo which operates the steering mechanism.

### Schematics

**Component Diagram**



**Component Table**

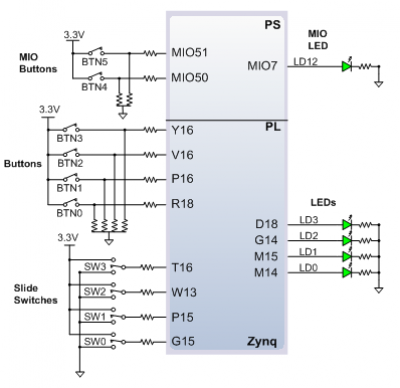
|  |  |  |  |
| --- | --- | --- | --- |
| **Callout** | **Component Description** | **Callout** | **Component Description** |
| **1** | Power Switch | **15** | Processor Reset Pushbutton |
| **2** | Power Select Jumper and battery header | **16** | Logic configuration reset Pushbutton |
| **3** | Shared UART/JTAG USB port | **17** | Audio Codec Connectors |
| **4** | MIO LED | **18** | Logic Configuration Done LED |
| **5** | MIO Pushbuttons (2) | **19** | Board Power Good LED |
| **6** | MIO Pmod | **20** | JTAG Port for optional external cable |
| **7** | USB OTG Connectors | **21** | Programming Mode Jumper |
| **8** | Logic LEDs (4) | **22** | Independent JTAG Mode Enable Jumper |
| **9** | Logic Slide switches (4) | **23** | PLL Bypass Jumper |
| **10** | USB OTG Host/Device Select Jumpers | **24** | VGA connector |
| **11** | Standard Pmod | **25** | microSD connector (Reverse side) |
| **12** | High-speed Pmods (3) | **26** | HDMI Sink/Source Connector |
| **13** | Logic Pushbuttons (4) | **27** | Ethernet RJ45 Connector |
| **14** | XADC Pmod | **28** | Power Jack |

### Signal Information

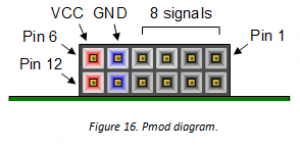
**Component Information**

|  |  |  |  |
| --- | --- | --- | --- |
| **Supply** | **Circuits** | **Device** | **Current (max/typical)** |
| 3.3V | FPGA I/O, USB ports, Clocks, Ethernet, SD slot, Flash, HDMI | IC26#1: ADP5052 | 2.5A/0.1A to 1.5A |
| 1.0V | FPGA, Ethernet Core | IC26#2: ADP5052 | 2.5A/0.2A to 2.1A |
| 1.5V | DDR3 | IC26#3: ADP5052 | 1.2A/0.1A to 1.2A |
| 1.8V | FPGA Auxiliary, Ethernet I/O, USB OTG | IC26#4: ADP5052 | 1.2A/0.1A to 0.6A |
| 1.8V | XADC Analog | IC26#5: ADP5052 | 200mA/20mA |
| 3.3V | Audio Analog | IC6: ADP150 | 150mA/50mA |
| 1.25V | XADC Precision Reference | IC27: ADR127 | 5mA/50uA |

**Basic Input/Output Pinout**



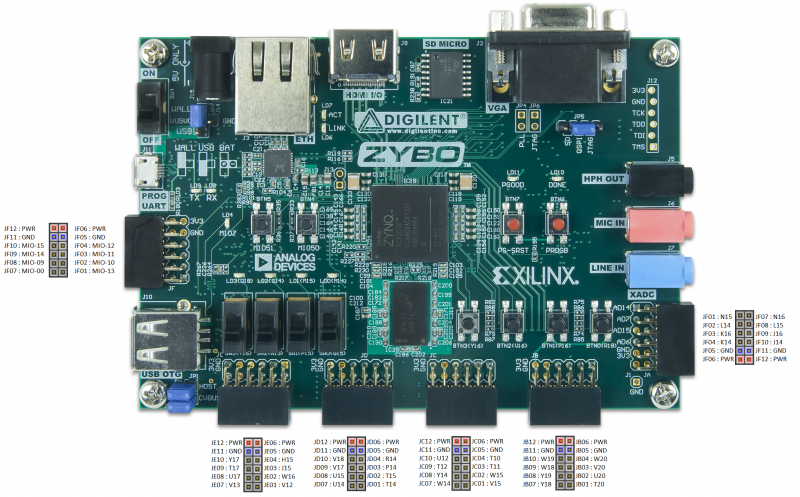
**Pmod Diagram**

****

**Pmod Pinouts**

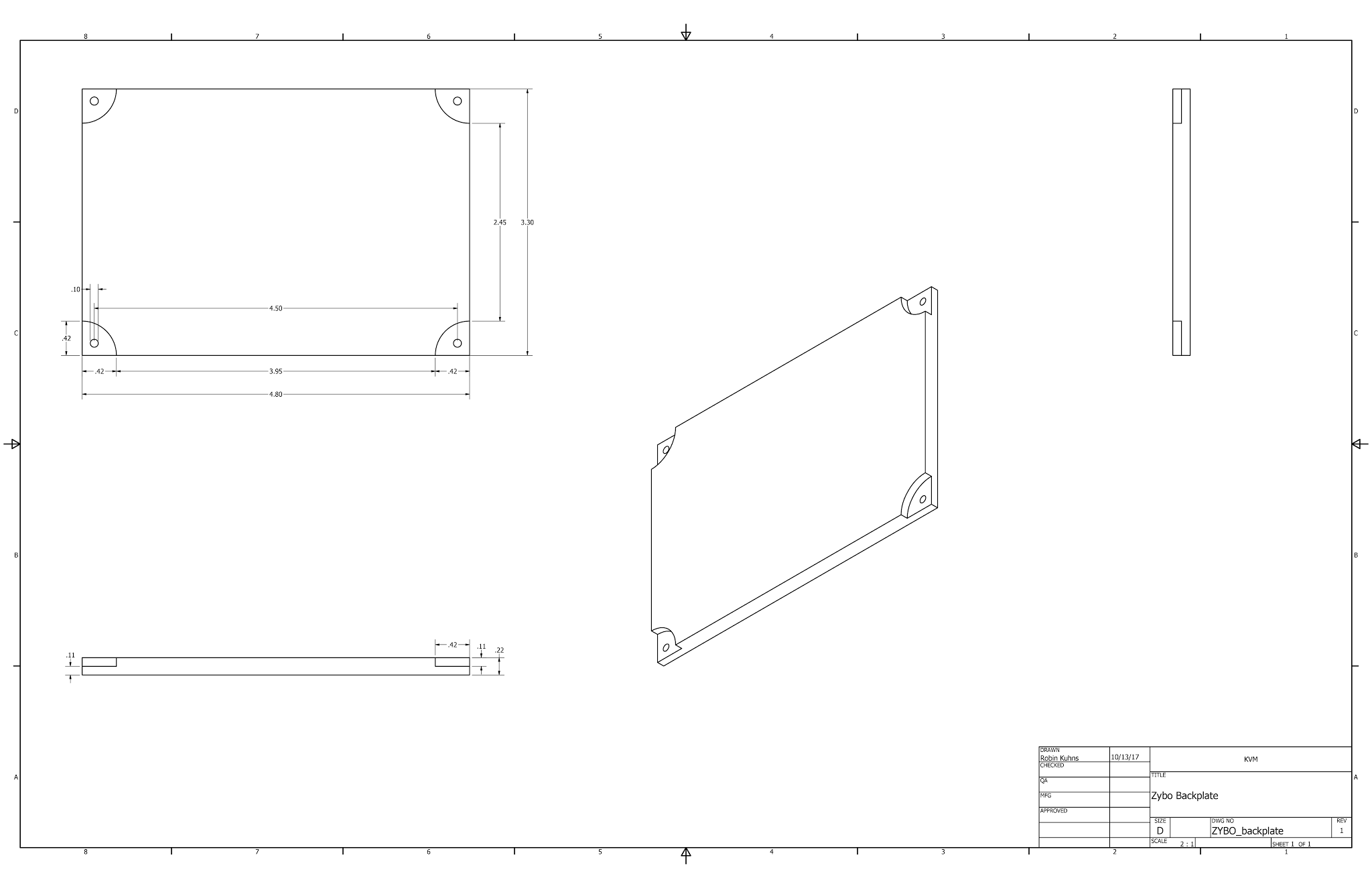
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pmod JA (XADC) | Pmod JB (Hi-Speed) | Pmod JC (Hi-Speed) | Pmod JD (Hi-Speed) | Pmod JE (Hi-Speed) | Pmod JF (MIO) |
| JA1: N15 | JB1: T20 | JC1: V15 | JD1: T14 | JE1: V12 | JF1: MIO-13 |
| JA2: L14 | JB2: U20 | JC2: W15 | JD2: T15 | JE2: W16 | JF2: MIO-10 |
| JA3: K16 | JB3: V20 | JC3: T11 | JD3: P14 | JE3: J15 | JF3: MIO-11 |
| JA4: K14 | JB4: W20 | JC4: T10 | JD4: R14 | JE4: H15 | JF4: MIO-12 |
| JA7: N16 | JB7: Y18 | JC7: W14 | JD7: U14 | JE7: V13 | JF7: MIO-0 |
| JA8: L15 | JB8: Y19 | JC8: Y14 | JD8: U15 | JE8: U17 | JF8: MIO-9 |
| JA9: J16 | JB9: W18 | JC9: T12 | JD9: V17 | JE9: T17 | JF9: MIO-14 |
| JA10: J14 | JB10: W19 | JC10: U12 | JD10: V18 | JE10: Y17 | JF10: MIO-15 |

**Pmod Pinouts In Relation To Board**

****

### Mechanical Designs

### Acrylic Backplate



### Module Function Specification

The board will have multiple communication interfaces controlled using the pinout available on the hardware itself. PWM signals will be sent to the servos of the steering mechanism, the pan-tilt mechanism of the Pixy Cam, and the motors driving the wheels. 5V and ground will be sent to the line scan camera and LIDAR sensor, which will then in return generate signals received at the pins of the board to use in its various logic statements to control the direction and speed of the vehicle. In manual mode, all of these logic statements are left unused, and the web interface communicates with the Bluetooth module of the vehicle to give the user full control over the direction, speed, and camera angle.

## FRDM Micro-Controller

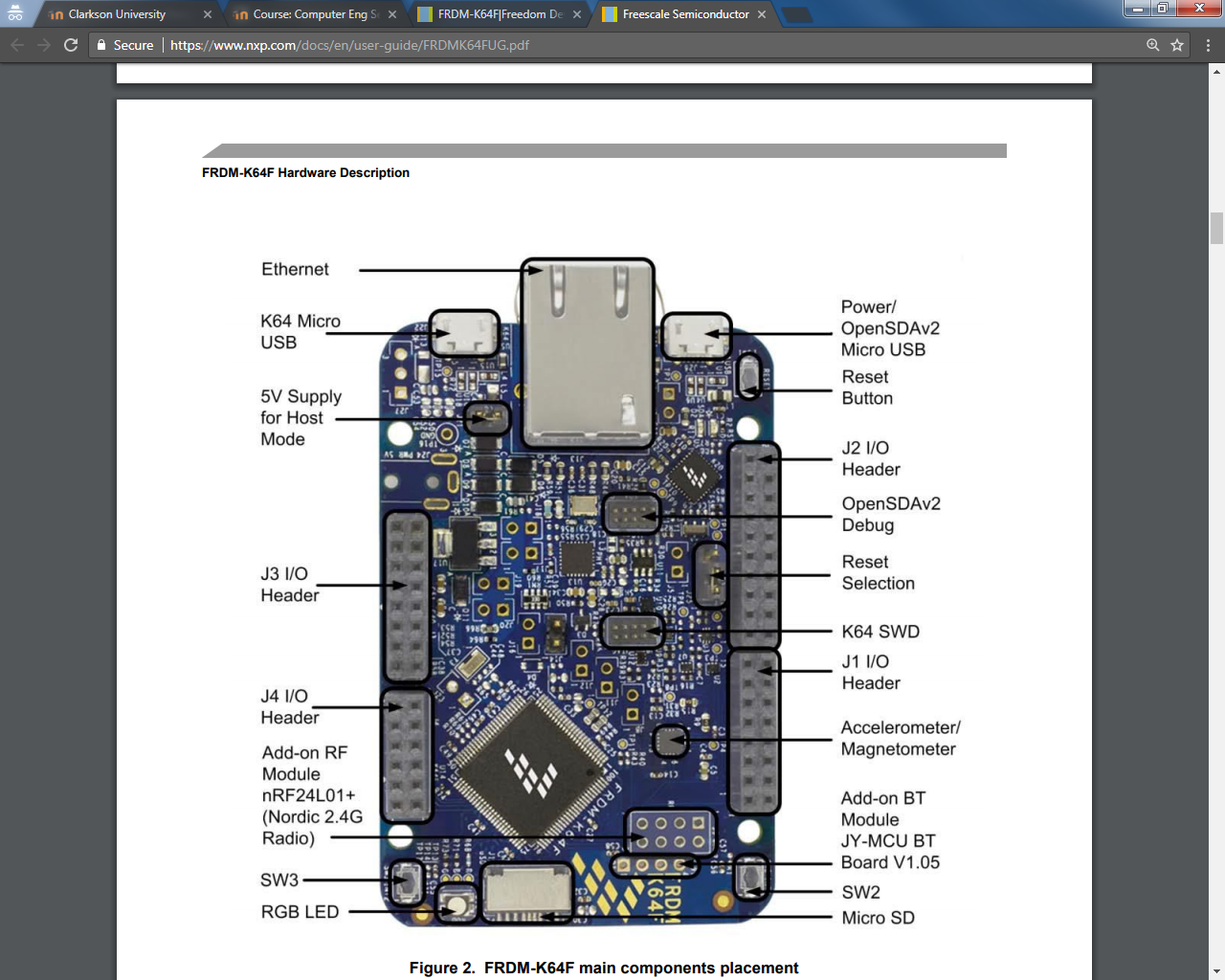
### Part Name

FRDM-K64F

### Functional Specification

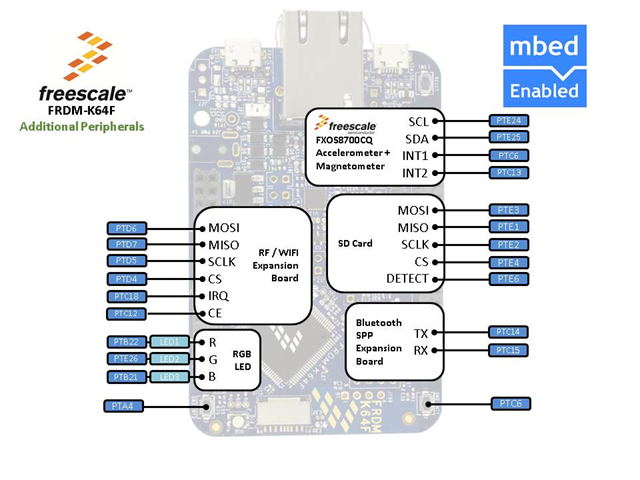
Act as the primary logic and control unit for the chassis, with multiple communication interfaces for interaction with sensors to receive data. These sensors include a Bluetooth module for manual input, a CMUcam5 Pixy Cam for object recognition, a TAOS line scanner for track recognition, and an LIDAR sensor for collision avoidance. The board also transmits signals to an H-Bridge motor driver which drives the wheels, a servo mechanism which controls the pan and tilt of the Pixy Cam, and another servo which operates the steering mechanism.

### Schematics



### Signal Information





### Mechanical Designs

### Acrylic Backplate

### FRDM_backplate.png

### Module Function Specification

The board will have multiple communication interfaces controlled using the pinout available on the hardware itself. PWM signals will be sent to the servos of the steering mechanism, the pan-tilt mechanism of the Pixy Cam, and the motors driving the wheels. 5V and ground will be sent to the line scan camera and LIDAR sensor, which will then in return generate signals received at the pins of the board to use in its various logic statements to control the direction and speed of the vehicle. In manual mode, all of these logic statements are left unused, and the web interface communicates with the Bluetooth module of the vehicle to give the user full control over the direction, speed, and camera angle.

## Visual Sensor

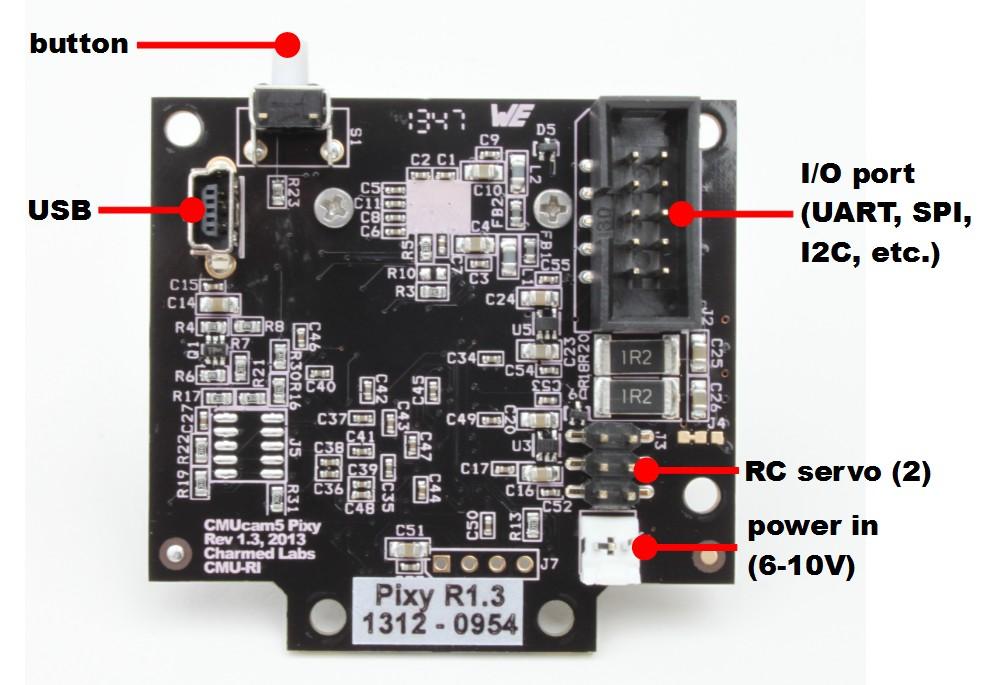
### Part Name

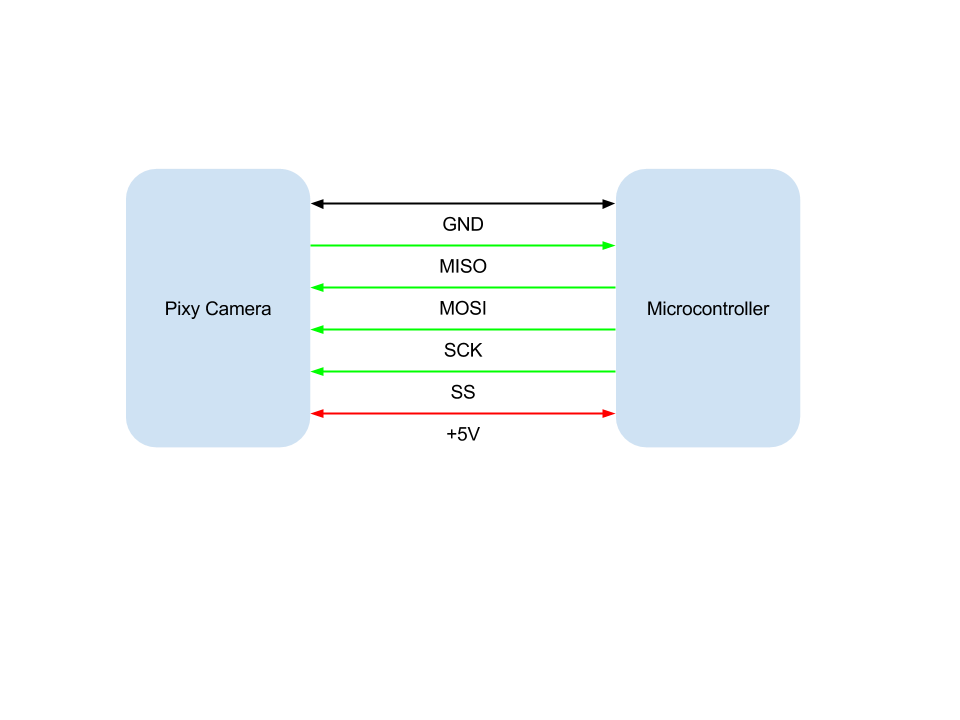
Cmucam5 (Pixy)

### Functional Specification

Recognize predefined objects based on light reflection frequency and proximity to other detected objects and transmit relative location and orientation data to the microprocessor unit using an SPI data link. The information sent by the camera will be used to determine the direction taken at an intersection in the track at runtime.

### Schematics

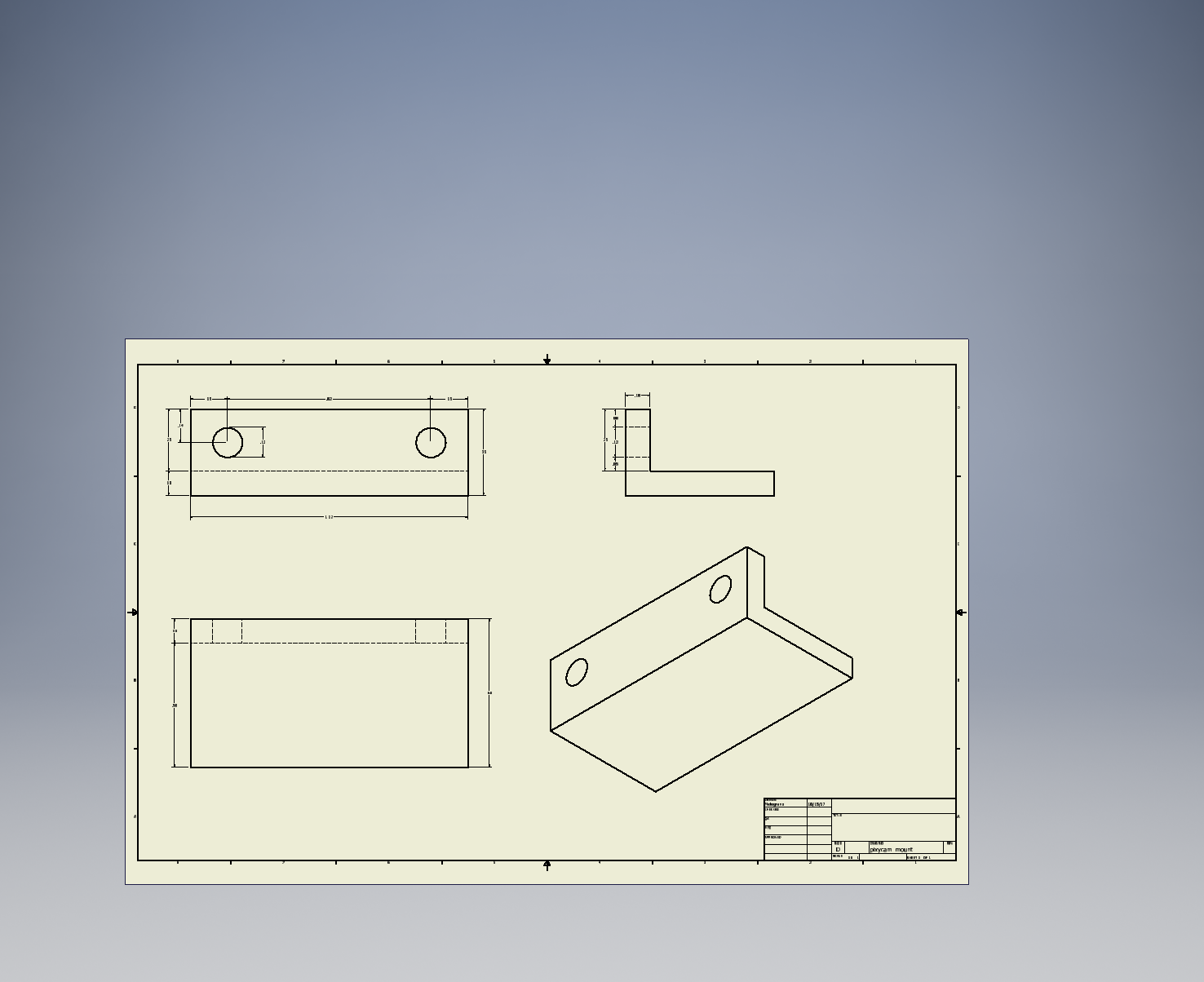




### Signal Information

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Wire Label** | **Pin Direction** | **Voltage** | **Current** | **Timing** | **Purpose** |
| +5V | N/A | 5v | 140mA | Constant | Power |
| GND | N/A | 0 v | 140mA | Constant | Ground |
| MISO | Out | 3.3 v | N/A | N/A | Data Out |
| MOSI | In | 3.3 v | N/A | N/A | Data In |
| SCK | In | 3.3 v | N/A | 48MHz | Clock |
| SS | In | 3.3V | N/A | N/A | Slave Select |

### Mechanical Designs



The right angle bracket depicted above will be used to mount the Pixy Camera onto the car. It will be 3D printed out of ABS. Using CA glue, the bracket will be mounted to the chassis.

### Module Function Specification

The function of the Pixy camera is to provide the information necessary to the microprocessor to make the proper turn at an intersection.

## Line Scan Camera

### Part Name

TSL1401

### Functional Specification

The Line Scan camera captures a 1 x 128 pixel image that will allow the car to see where in relation to the line it is. This knowledge will allow the car’s steering to make micro adjustments to stay on the track.

### Schematics

### Line Scan Camera.png

### 

### 

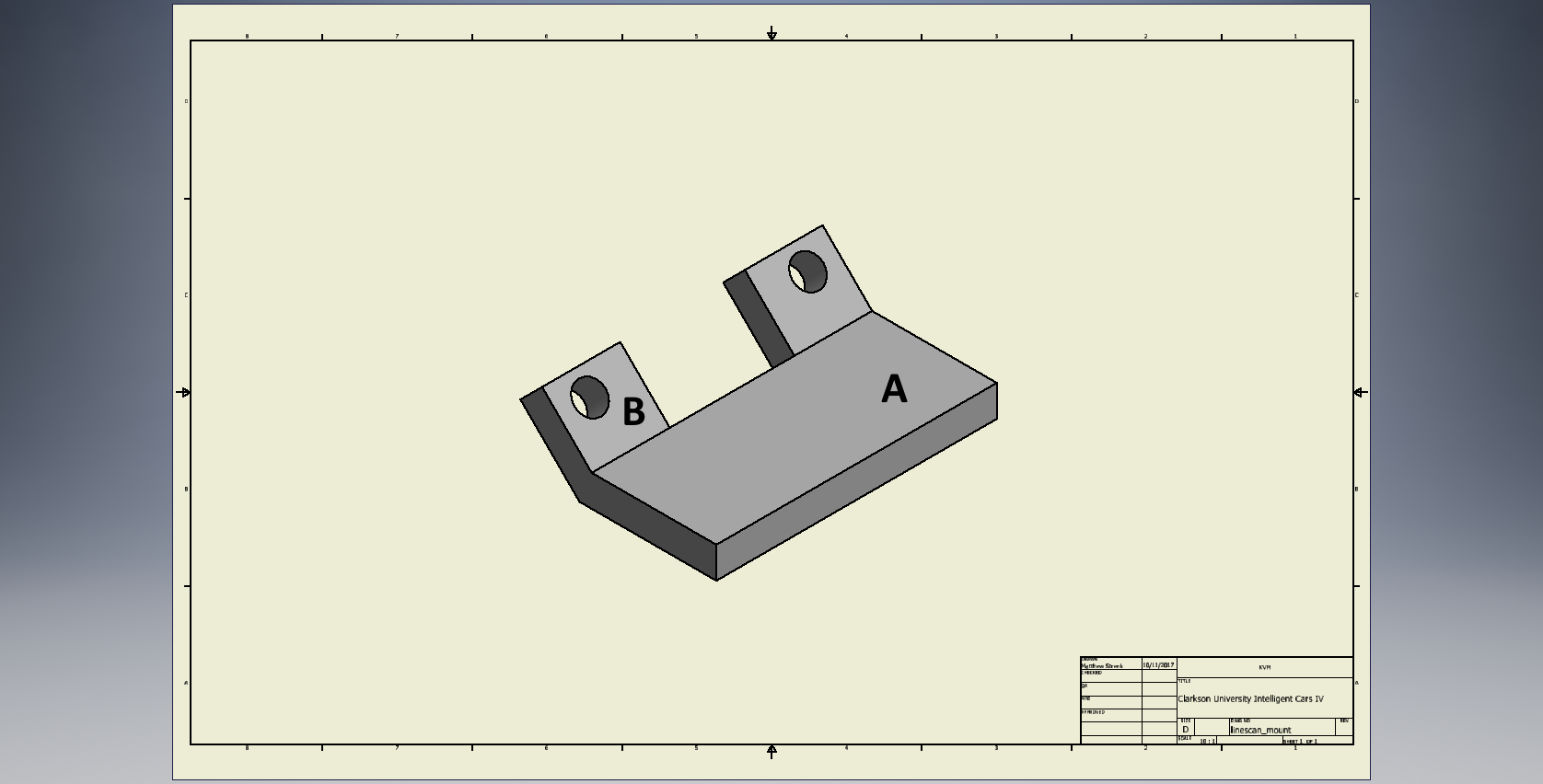
### 

### Signal Information

### 

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Wire Label** | **Pin Direction** | **Voltage** | **Current** | **Timing** | **Purpose** |
| Vdd | In | 3.3 v | 2.6 mA | Constant | Power |
| Ground | In | 0 v | 0 A | Constant | Ground |
| Clk | In | 3.3 v | N/A | 6.45 kHz | This is the clock for the camera. The faster the clock is the faster the pixels are released for another capture. |
| SI | In | 3.3 v | N/A | 1 input pulse high at beginning of capture cycle, low at any other time. PWM Frequency will be 50 Hz. | Serial Input to Sensor. Triggers the start of a capture. The closer each pulse is, the faster the captures happen. |
| AO | Out | 0 v - 3.3 v | 25 mA | Occurs for 129 clock cycles | Analog representation of the 128 pixels that are captured by the camera. |

### Mechanical Designs



The figure above shows the isometric diagram for the Line Scan camera. This will mount the camera to the front of the car at a 45 degree angle. There is a known issue with the Line Scan camera where the light that shines through the PCB interferes with the charging of the pixels. To fix this problem, a piece of electrical tape will be placed on the back of the camera’s PCB.

### 

|  |  |
| --- | --- |
| **Part** | **Function** |
| A | This is the part of the mount that will adhere to the front of the car using glue. |
| B | This part of the mount is where the Line Scan Camera will attach to the mount. The mount has two ⅛ in diameter holes that will match up with the two ⅛ in diameter hole on the Line Scan Camera. There is a 135 degree angle between the base of the mount(A) and mount holes(B). This will allow the camera to see ahead of the car. |

### 

### linescan_mount-1.png

The diagram above shows the front, side, top, and isometric views of the mount.

### 

### Module Function Specification

### Linescan integration time.PNG

Using the Line Scan camera’s two data inputs, the clock (clk) and the serial input (SI) can change the frame speed of the camera. The faster the clock is, the faster the camera will release the pixel values to the microcontroller. The closer each of the SI pulses are to each other, the faster the captures will occur. Although the faster that the captures occur, the less the pixels will charge before they are released by the camera. This can cause the images to be too dark and therefore the information would be useless.

When the serial input pulses the Line Scan camera starts new capture. After the capture starts the clock will run for 129 cycles to capture the 1 x 128 pixel resolution image. After the 129 cycles have completed, the clock will stop for one clock cycle to allow for the pixels to charge uninterfered. When the serial input pulse is sent again, the capture will start again.

## H-Bridge

### Part Name

MDD10A Dual​ ​Channel​ ​10A​ ​DC​ ​Motor​ ​Driver

### Functional Specification

Act as the interface between the microcontroller and the motors. Motor controls are done using the PWM inputs of this board. An H-bridge is used to vary the power output to the motors.

### Schematics



### 

### 

### 

### Signal Information

|  |  |
| --- | --- |
| **Pin** | **Description** |
| GND | Ground |
| PWM2 | PWM speed input for motor 2 |
| DIR2 | Direction input for motor 2 |
| PWM1 | PWM speed input for motor 1 |
| DIR1 | Direction input for motor 2 |
| M1 B | Connection for motor 1 terminal B |
| M1 A | Connection for motor 1 terminal A |
| Power + | Positive battery terminal |
| Power - | Negative battery terminal |
| M2 A | Connection for motor 2 terminal A |
| M2 B | Connection for motor 2 terminal B |

### Mechanical Designs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Min** | **Typical** | **Max** | **Unit** |
| VCC (Power Input Voltage) | 5 | - | 30 | V |
| IMAX (Maximum Continuous Motor Current) | - | - | 10 | A |
| IPEAK (Peak​ ​Motor​ ​Current) | - | - | 30 | A |
| VIOH (Logic​ ​Input​ ​High​ ​Level) | 3 | - | 5.5 | V |
| VIOL (Logic​ ​Input​ ​Low​ ​Level) | 0 | 0 | 0.5 | V |
| fMAX(Maximum​ ​PWM​ ​Frequency) | - | - | 20 | KHz |

### 

### 

### 

### Module Function Specification

|  |  |  |  |
| --- | --- | --- | --- |
| **PWM In** | **DIR In** | **Terminal A Out** | **Terminal B out** |
| Low | X | Low | Low |
| High | Low | High | Low |
| High | High | Low | High |

## Voltage Regulator Module

### Part Name

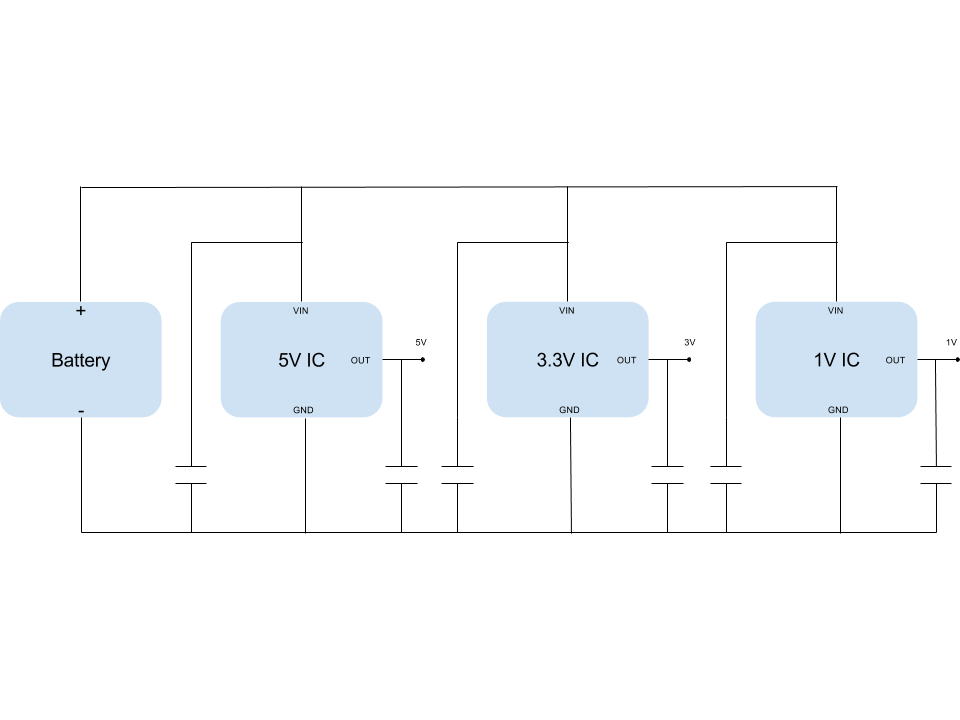
Custom; Designed using:

1. TDK Corporation FG24X5R1E226MRT06
2. Diodes Incorporated AZ1084T-5.0E
3. Rohm Semiconductor BA33DD0T
4. ON Semiconductor NCV571SN10T1G

### Functional Specification

This regulator module takes the battery voltage and converts it into 3 outputs, 1V, 3.3V, and 5V. These drive the boards, DAC, cameras, and LEDs.

### Schematics



*All capacitors in the circuit above are 22uF.*

### 

### 

### 

### Signal Information

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Min** | **Typical** | **Max** | **Unit** |
| VCC (Power Input Voltage) | 6.4 | 7.4 | 8.4 | V |
| IOUT ( 5V Output Current) | .5 | 1.5 | 5 | A |
| IOUT ( 3.3V Output Current) | 250 | 1 | 1.5 | mA |
| IOUT ( 1V Output Current) | 100 | 150 | 200 | mA |

### Mechanical Designs [Design Phase 2]

### Module Function Specification

The outputs need to be within 0.1V of the desired values in order to avoid damaging the receiving components, the regulator chips were chosen to be well within this specification.

## LIDAR [Design Phase 2]

### Part Name

### Functional Specification

### Schematics

### Signal Information

### Mechanical Designs

### Module Function Specification

## Bluetooth [Design Phase 2]

### Part Name

### Functional Specification

### Schematics

### Signal Information

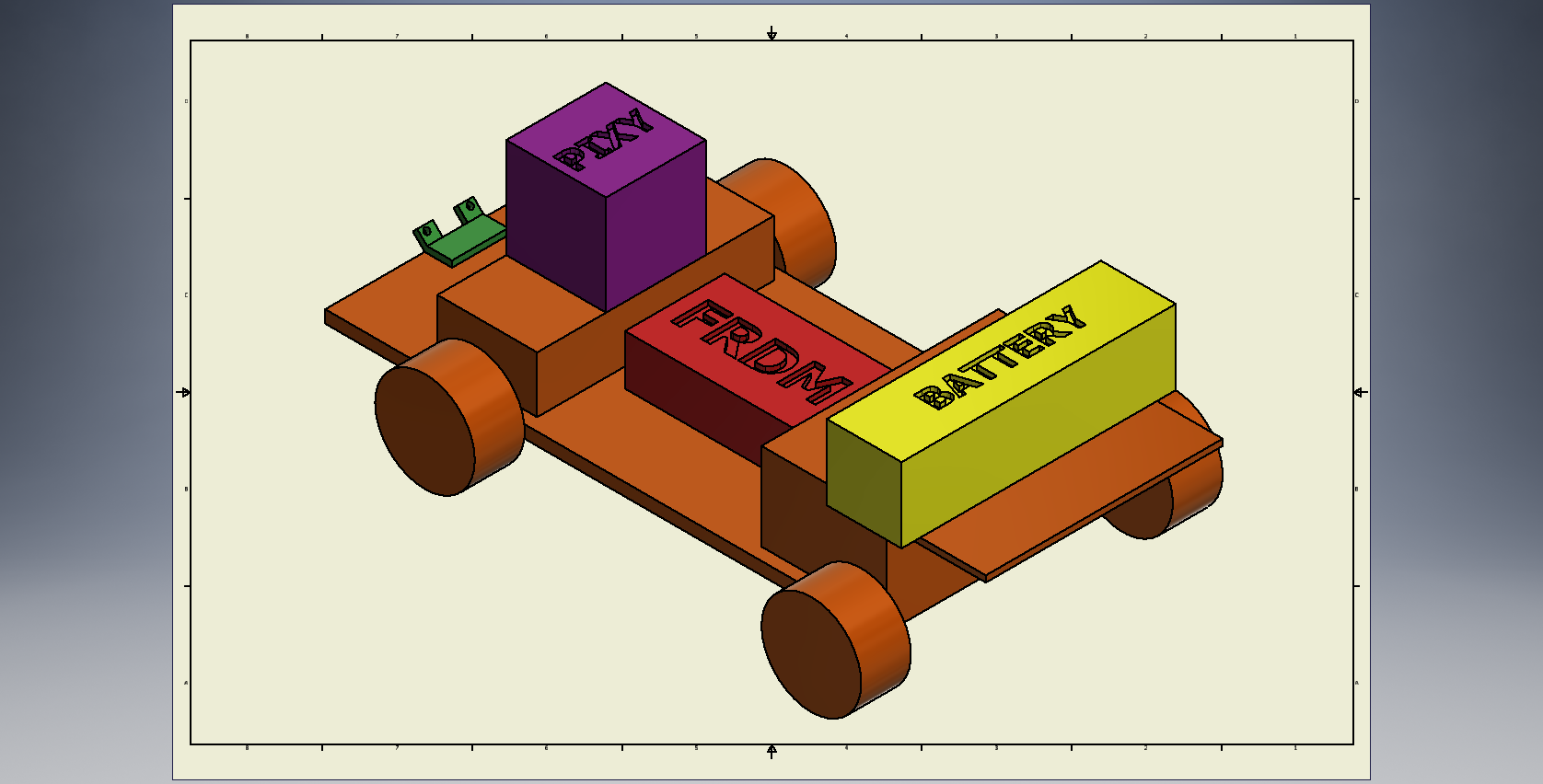
### Mechanical Designs

### Module Function Specification

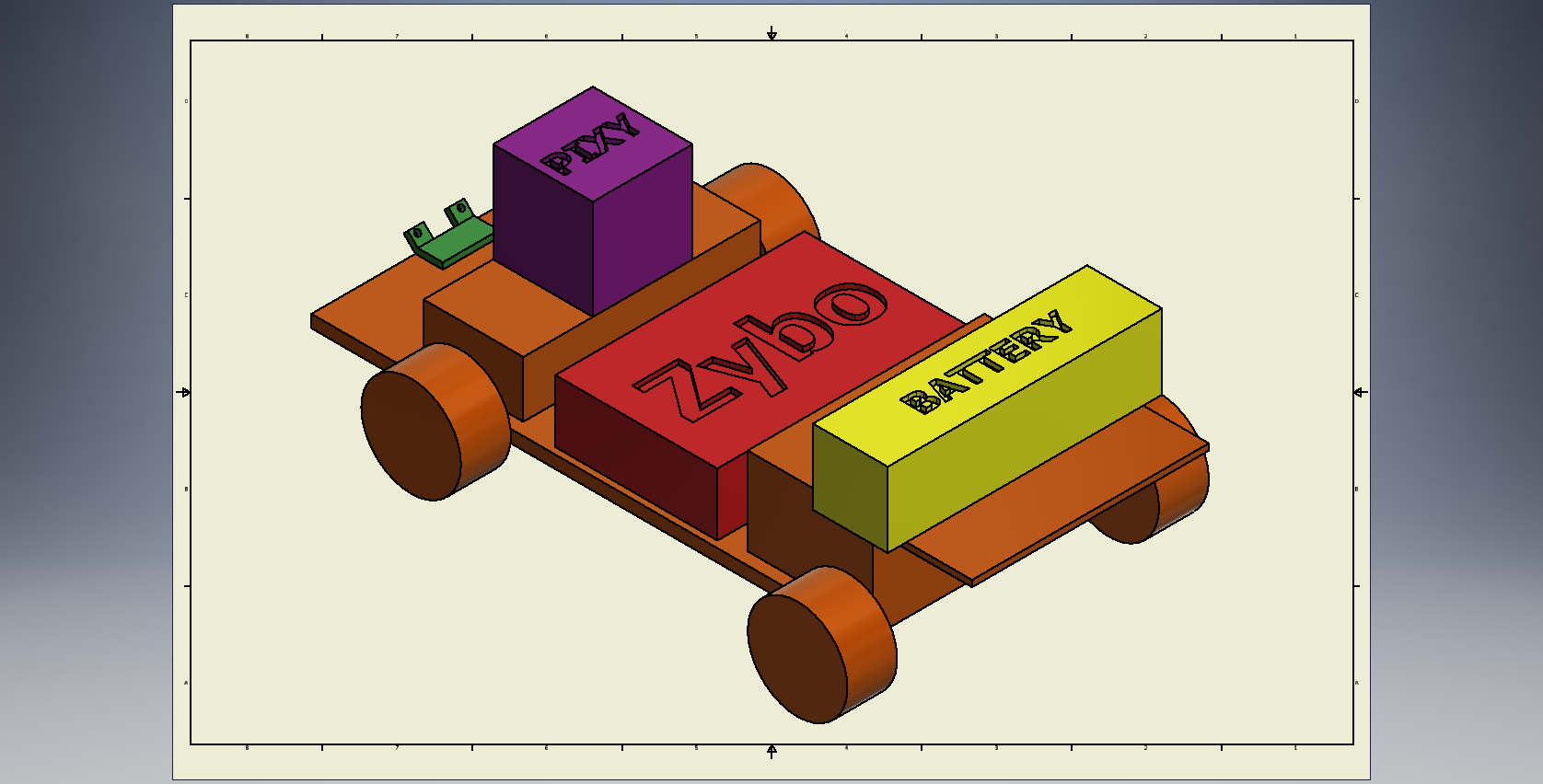
# Mechanical Detailed Design

## Full Car Design

* + 1. **Functional Specification** This is a rough sketch of where all of the components will live on the car.
    2. **Schematics and Isometric Diagrams** The components that are shown in the diagram below are the base of the car, either the FRDM or Zybo microcontroller board, the battery, the mount for the Line Scan Camera, and the mount for the Pixy Camera and the pan/tilt mechanism.

****

Above is a rough sketch of where each of the components will live on the car for the FRDM board.

****

Above is a rough sketch for where each of the components will live on the car for the Zybo board.

# References

* 1. KVM, *System Requirements Specifications Document*. Clarkson University, 2017.
  2. *FRDM-K64F Freedom Module User's Guide*. NXP Semiconductors, 2017.
  3. *ZYBO FPGA Board Reference Manual*. Digilent, 2017.
  4. IEEE Standard for System and Software Verification and Validation," in *IEEE Std 1012-2012 (Revision of IEEE Std 1012-2004)* , vol., no., pp.1-223, May 25 2012
  5. IEEE Standard for Local and metropolitan area networks--Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)," in *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)* , vol., no., pp.1-314, Sept. 5 2011
  6. IEEE Guide: Test Procedures for Direct-Current Machines," in *ANSI/IEEE Std 113-1985* , vol., no., pp.1-38, Dec. 21 1984
  7. *TSL1401CL Datasheet.* AMS AG, 2011
  8. G. Xiaoli, “Line Scan Camera Use,” NXP Community, 25-Mar-2015. [Online]. Available: https://community.nxp.com/docs/DOC-1030. [Accessed: 14-Oct-2017].