HCS12 Autonomous Robotic Car

Prepared for
Professor Peter H.
DeVry ECET 365

Prepared by
Derek Scarborough

25 Feb 14

CONTENTS

PAGE

INTRODUCTION/PROJECT DESCRIPTION 1

PURPOSE, SCOPE, AND LIMITATIONS 1
SOURCES AND METHODS 1
REPORT ORGANIZATION 1
HARDWARE BLOCK DIAGRAM 2
INPUT/OUTPUT MAP 2
INPUT/OUTPUT SPECIFICATION 3
HARDWARE SPECIFICATION 3
HARDWARE THEORY OF OPERATION 4
VISUAL SUBSYSTEM 4
MOTOR SUBSYSTEM 4
POWER SUBSYSTEM 4
STATUS SUBSYSTEM 4
HARDWARE CONSTRUCTION 5
TRACK CONSTRUCTION 5
SOFTWARE DOCUMENTATION 5
SOFTWARE THEORY OF OPERATION 6

USER'S MANUAL 7

FEATURES AND BENEFITS 7
POWER-UP 7
INSTRUCTIONS 8

SYSTEM DOCUMENTATION 8

CAR.C 8

APPENDIX A (DATA SHEETS) 12

APPENDIX B (ILLUSTRATIONS) 13

APPENDIX C (CONSTRUCTION FIGURES) 14

REFERENCES 17

List of Illustrations

FIGURES PAGE

1) Hardware Block Diagram.....2

2) Power Theory Diagram.....Appendix B

3) Robotic car track.....5

4) Flowchart.....6

5a) Powering the car.....7

5b) Resetting the car.....7

5c) Replacing batteries.....7

HCS12 Autonomous Robotic Car

Description

This report entails procedures, block diagrams, I/O maps, I/O definition, hardware theory, software documentation and/or flow chart and software theory for a constructed HCS12 autonomous robotic car. This report is designed to convey pertinent information in regards to the universality and real-world possibilities of the HCS12 and its use as portrayed in this report as an autonomous controller. This report will discuss the use of transducers (optical sensors) and their reading of specified systems to direct and control the operation autonomously of the car.

Purpose, Scope, and Limitations

The purpose of this report is to explain, analyze, and instruct the use of the HCS12 robotic car. An HCS12 can be (and has been) constructed to operate and control an autonomous robotic car. For example, the HCS12 can be programmed to control speed of car via it's internal PWM (pulse width modulation). It also can be coded to know when to stop, turn, reverse, etc. This report does not warrant or guarantee success in HCS12 autonomous design. These variables are independent and varies to each engineer's specific creativity. What this report does achieve is a succinct demonstration of the construction, programming, and demonstration of the HCS12 as the controller for an autonomous robotic car.

Sources and Methods

The sources and methods used to gather the enclosed information were acquired via multiple tomes. The code was written with CodeWarrior. The HCS12 car was built with the HCS12 tower system. Ancillary information was acquired from DeVry ECET 365 weeks' 1 through 7 course and lectures. Also, two major tomes were used to assist in development: *Microcontrollers and Embedded Systems (*Mazidi, 2009) and *Embedded Microcomputer Systems (*Valvano, 2012). Also, the PmodHB5, PmodLS1, IG-22GM, OPB704, TWR-Breadboard, and HCS12 data sheets were utilized.

Report Organization

This report will begin with a hardware block diagram. Next, an I/O map will be discussed. Following that will be I/O definition. Hardware theory of operation will be next. Then software and / or a flowchart will be discussed. There will be a discussion on software theory of operation. A user's manual of car operation will be introduced. Any relevant hardware and software support documentation will be given. Troubleshooting will be addressed as necessary. The report will conclude with an appendix.

Hardware Block Diagram

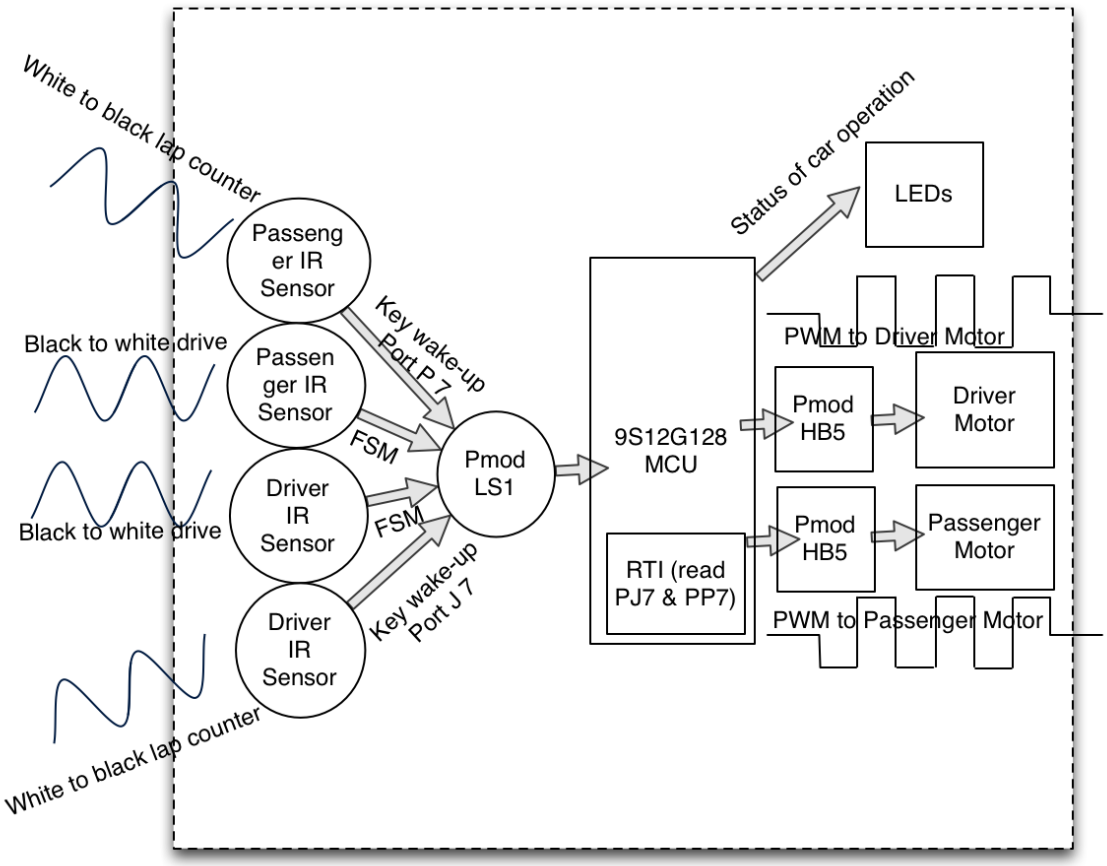


Figure 1: Hardware block diagram

Input / Output Map

Input:

IR Sensors (x4): IR Sensor (Passenger) - Lap counter
IR Sensor (Passenger) - FSM routine for operation and control
IR Sensor (Driver) - FSM routine for operation and control
IR Sensor (Driver) - Lap counter

PmodLS1: Converts IR sensors to digital inputs

Output:

LEDs: PTT, Pins 7:4, Outputs state of motors (straight, right, left, stop)
PWM: 250 Period / 171 Duty cycle to (2 x DC motors)
PmodHB5 (x2): H-bridge driver for small/medium sized DC motor
IG-22GM DC Motor (x2): DC geared motor to operate car wheels

Input/Output Specfication

Reflective object sensor
- Optek Technology OPB704
- Input diode: Forward DC current 40mA
  - Reverse DC voltage 2V
  - Power dissipation 100mW
- Output Photodetector: Emitter-Collector voltage 5V
  - Collector DC Current 25mA
  - Power dissipation 100mW

PmodLS1: 5V operation
- Input: OPB704 IR sensors (x4)
- Output: Logical 0/1 (0 for black/1 for white)

PmodHB5: Motor operation up to 12V
- H-bridge operation 5V
- Input: Enable via PWM
  - Direction via PortB
- Output: On/Off via Enable
  - Direction control
  - PmodHB5 cabling

IG-22GM DC Motor: 12V operation

Hardware Specification

System: HCS12 Tower
- Primary/Secondary elevators
- HCS12 MCU board
- Tower breadboard: connectivity and component integration into HCS12 Tower

Power: 4 x Li-ion 18650 (14.8V)

Components parts:
- 2 x 1000µF capacitors
- 2 x 0.1 µF capacitors
- 1 x 100mH inductor
- 1 x 7805 5V voltage regulator
- 1 x 7812 12V voltage regulator

Chassis:
- 2 x wheels
- 3 x SPST toggle switches
- 1 x drag post
- 1 x wiring connect terminal
- Copper cabling for voltage and ground wiring

Hardware Theory of Operation

VISUAL SUBSYSTEM (INPUT)

The operation of the autonomous car, largely, depends on the input of the 4 optical sensors. Two middle sensors (black to white) for driver and passenger dictate control of the car. If both sensors are on black, the car moves forward. If driver sensor reflects white, only the driver motor rotates while passenger motor stops until both are logical 0 (on black). The reverse holds true for passenger; if passenger sensor reflects white, passenger motor spins while driver motor stops until logical 0 for both middle sensors.

The two outer sensors are used as part of key wake-up interrupts which is fed into an RTI. If both sensors (one is Port J interrupt 24, one is Port P interrupt 56) report logical 0 (black) the RTI increases count by one. Once count becomes two, the vehicle stops. Both outer sensors happen simultaneously, although Port J will happen first due to higher priority. The RTI is constantly checking at about 100ms.

The optical sensors all send logical 0/1 to the MCU via the PmodLS1. The PmodLS1 is able to convert analog optical input to digital input signals before sending to HCS12.

MOTOR SUBSYSTEM (OUTPUT)

The HCS12 is capable of providing digital energy to the PmodHB5. The PmodHB5 takes this digital energy and drives its the two DC motors. The PWM chosen for this car is 250 period with a 68.5% DC of 171 duty cycle. The motor requires approximately 328mA of current which is provided via the PWM to the PmodHB5 to the motors.

POWER SUBSYSTEM

The power subsystem consists mainly of 4 lithium-ion batteries at 3.7V/3600mAh each totaling 14.8V. The 14.8V is divvied up between two voltage regulators. One regulator is the 7812 supplying an average of 10.84V - 11.89V to DC motors. A 12V regulator was chosen to ensure voltage supplied did not exceed max voltage to DC motor of 12V. The 2nd regulator is the 7805. The 7805 supplies a regulated voltage average of 5.05V to the HCS12 (refer to Appendix B figure 2). This is below the max of 5.5V.

STATUS SUBSYSTEM (OUTPUT)

The 4 LEDs indicate what operation is in effect. The LEDs correlate to current PWME value state. If 0x0A (both motors on), the LEDs should be LED4 and LED2. If 0x08 (driver motor on), LED4 on. For 0x02 (passenger motor on), LED2 on. And for 0x00 (both motors off), all LEDs on.

Hardware Construction

Aside from the standard directions of construction, various creative decisions were implemented. A Dremel rotary tool was used to create alignment of tower elevator holes with chassis. The Dremel was also used to create large enough holes for 3 SPST switches. Soldering of ground to connect terminal to ensure solid ground for entire system was also implemented. Refer to Appendix C for construction illustrations.

Track Construction

The robotic car track was created with 2 40x30 foam boards. The boards were combined to form an 80x60 track. The track itself was created with tacks and string. The oval was initially traced with pencil. Blue painters tape was then overlaid over the pencil trace. Finally, black gorilla duct tape was placed over the painters tape.

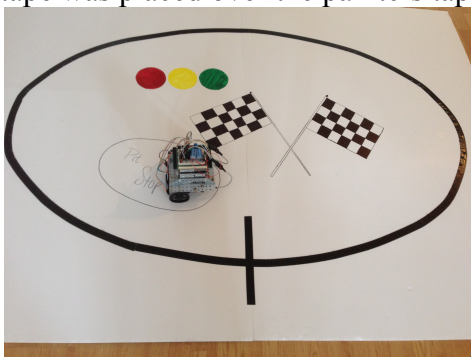


Figure 3: Robotic car track (80x60)

Software Documentation

High-level block diagram

car.c
derivative.h
mc9s12g128.h

Type of files

*C language used for all files (inline assembly for enable interrupts)
*Freescale CodeWarrior IDE used for programming environment
*car.c functions:

```
void init_ISR(void);        //set up ISRs
void init_PORTS(void);      //set up Ports
void init_PWM(void);        //set up PWM
```

Flowchart

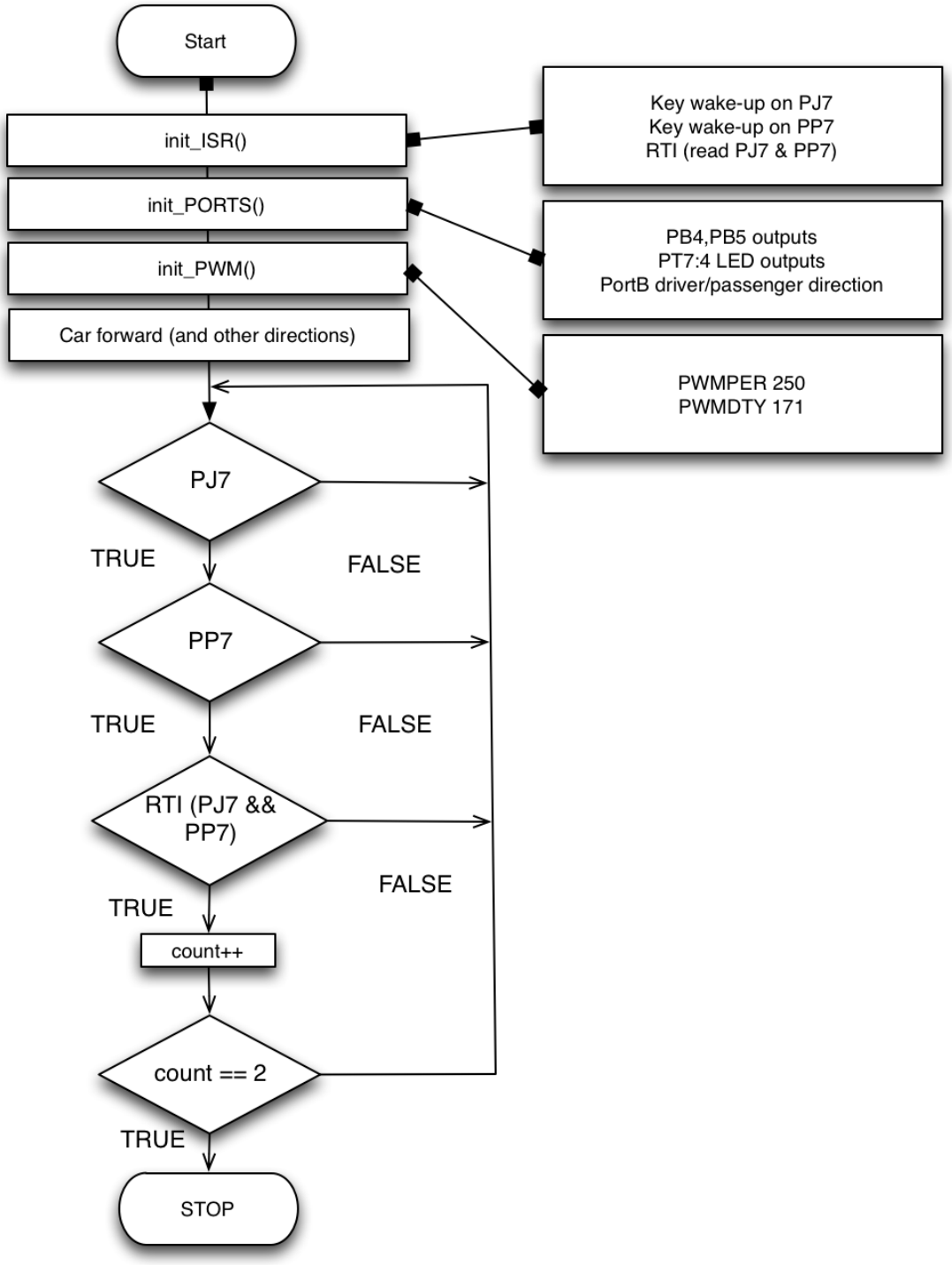


Figure 4: Flowchart

Software Theory of Operation

The software for the car uses two semaphores (flags) and one count variable to control lap count. The PWM does not vary during operation and is set at 68% duty cycle. Initialization begins with the following 3 functions:

- init_ISR()
  This function sets up the two key wake-up interrupts and RTI.
- init_PORTS()
  This function initializes basic ports used by the MCU. It also initializes the status LEDs for status of car.
- init_PWM()
  This function initializes the car to its designated duty cycle and thus its speed.

After initialization, interrupts are enabled (asm cli) and a for loop is entered until laps around reach 2. Key wake-up PJ7 is set anytime it crosses black and a flag is subsequently set. Simultaneously, if the black crossing is valid, key wake-up PP7 is to be triggered and its flag is set. If both PJ7 and PP7 flags are set, they are utilized by the RTI. The RTI is constantly checking for both flags to be set. When this condition is met, the RTI increments count by one (count++) and clears its flag and PJ7 and PP7 flags. When this condition is met a second time, the count increments to two and the car stops.

USER'S MANUAL

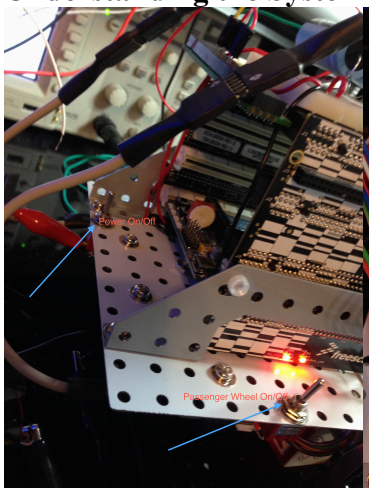
Features and Benefits

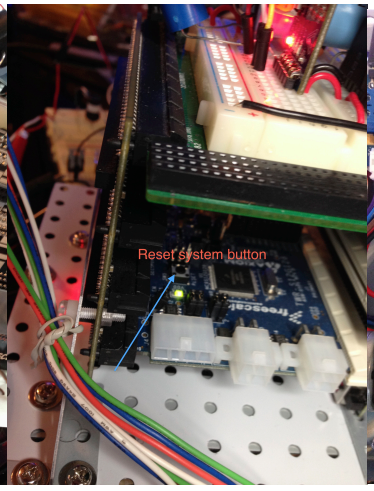
This HCS12 robotic car demonstrates MCU power and how code and hardware interact together. This car is representative of autonomous vehicles, such as the Google car, at a scaled down version. Additional benefits include wireless implementation for communications with other MCUs and architectures, control via computer, and even adding proximity sensors to avoid objects. Status of operation is given via LEDs.

Power-up

To turn on system, simply toggle power switch on/off. Also to ensure memory, RTI registers et. al are cleared, it is recommended to push reset button as well.

Understanding the System





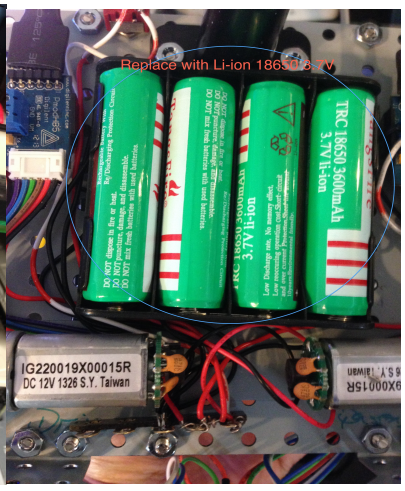


Figure 5a: Powering car *Figure 5b: Resetting car* *Figure 5c: Replacing batteries*

5A. **Powering car** via toggle switch
5B. **Resetting car** via HCS12 reset button
5C. **Replacing batteries** with similar Li-ion 18650 3.7V

Instructions
To power-up, toggle power switch as shown above to the left for on. Hit reset button to ensure clear memory and registers. For best operation, ensure both middle (driver and passenger) sensors are both off and in line with black tape.

SYSTEM DOCUMENTATION

car.c:

```
/*****************************************************************
 * *Programmer:        Derek Scarborough
 * *
 * *23 Feb 14
 * *
 * *ECET365 Professor Peter Han
 * *
 * *Assignment: Course Project Autonomous Robotic Car
 * *
 * *File name: car.c
 * *
 * *Description: This program causes the robotic car to perform two laps of track and
 * * stop after two laps
 * *
 * *Input: IR sensor via PmodLS1 to HCS12
 * *
 * *Output: PWM to motors via PmodHB5 and LEDs for status
 * *
*****************************************************************/
#include <hidef.h>          /* common defines and macros */
#include "derivative.h"      /* derivative-specific definitions */

void init_ISR(void);
void init_PORTS(void);
void init_PWM(void);

#define Driver 0x01
#define Passenger 0x02

#define Driver_On 0x08
#define Passenger_On 0x02

#define Driver_DIR 0x20
#define Passenger_DIR 0x10
```

```
unsigned char sensor;          // sensor for the inputs
unsigned char flagCheckered;   // Port P 7 sensor; passenger side
unsigned char checkeredFlag;   // Port J 7 sensor; driver side
unsigned char count;

const struct State
{
 unsigned char PWME_value;     // Output data to PWME
 unsigned char stateLED;
 const struct State *Next[4];  // Next state if input = 0,1 */
};

#define S0 &fsm[0]
#define S1 &fsm[1]
#define S2 &fsm[2]
#define S3 &fsm[3]

typedef const struct State StateType;

StateType fsm[4]=
{
 {0x0A,0x10,{S0,S1,S2,S3}}, // S0 means both motors should move
 {0x08,0x20,{S0,S1,S2,S3}}, // S1 means drive motor moves only
 {0x02,0x40,{S0,S1,S2,S3}}, // S2 means passenger motor moves only
 {0x00,0x80,{S3,S3,S3,S3}}  // S3 means both motors should stop
};


void main(void)
{
 //initial states
 StateType *Pt;         // pointer to present state
 Pt = S0;
 count = 0;
 checkeredFlag = 0;
 flagCheckered = 0;

 asm sei                //disable interrupts
 init_ISR();
 init_PORTS();
 init_PWM();
 asm cli                //enable interrupts

 for(;;)
 {
```

```
    sensor = PORTA & 0x03;      // bit 0 is for driver, bit 1 for passenger
    Pt = Pt->Next[sensor];      // Move to the next state
    PWME = Pt->PWME_value;      // load PWME value to its register
    PTT &= ~Pt->stateLED;       // Turn the state LED on

    if (count == 2)
          PWME = 0x00;
  }
}

/***************** ISRs*******************/
#pragma CODE_SEG NON_BANKED
void interrupt 24 checkeredFlagISR(void)
{
  if (PIFJ & 0x80)
    checkeredFlag = 1;
  else
    checkeredFlag = 0;

  PIFJ = 0x80;
}

void interrupt 56 flagCheckeredISR(void)
{
  if (PIFP & 0x80)
    flagCheckered = 1;
  else
    flagCheckered = 0;

  PIFP = 0x80;
}

void interrupt 7 checkeredRTI(void)
{
  if  (checkeredFlag && flagCheckered)
  {
    count++;
    checkeredFlag = 0;
    flagCheckered = 0;
  }

  CPMUFLG_RTIF = 1;
}
#pragma CODE_SEG DEFAULT
```

```
void init_ISR()
{
 //RTI
 CPMURTI = 0xEF;
 CPMUFLG_RTIF = 1;
 CPMUINT_RTIE = 1;

 //Key wake-up interrupt PJ7
 DDRJ &= ~0x80;
 PPSJ |= 0x80;
 PERJ &= ~0x80;
 PIEJ = 0x80;
 PIFJ = 0x80;

 //Key wake-up interrupt PP7
 DDRP &= ~0x80;
 PPSP |= 0x80;
 PERP &= ~0x80;
 PIEP = 0x80;
 PIFP = 0x80;
}

void init_PORTS()
{
 DDRB |= 0x30;          // Let PB4, PB5 becomes OUTPUT
 DDRT = 0xF0;           // PORT T7-T4 output (LEDs)
 PTT = 0xF0;           // initialize with LEDs off
 PORTB |= Driver_DIR;    // Set driver motor direction
 PORTB &= ~Passenger_DIR;// Set passenger motor direction
 DDRA = 0x00;           // PORTA as inputs
}
void init_PWM()
{
 PWMCAE = 0x00;             // left aligned
 PWMCLK = 0x00;             // Ch. 0 - Ch. 1 source is clock A, ch.2 & ch 3 use clock B
 PWMCLKAB = 0x00;             // Use clock A and B respectively
 PWMPOL = 0x0A;             // initial HIGH output on ch. 1 and 3 (Use odd # registers)
 PWMPRCLK = 0x33;             // Clk A pre-scale = 8
 PWMCTL = 0x30;             // CON01 = '1' and Con23 = '1'
 PWMPER01 = 250;
 PWMPER23 = 250;
 PWMDTY01 = 171;             // 68.5% duty cycle
 PWMDTY23 = 171;             // 68.5% duty cycle
}
```

Appendix A Data Sheets

MC9S12G Family Reference Manual and Data Sheet, Rev.1.23 (2013)

Digilent PmodHB5 2A H-Bridge Reference Manual, Rev.D (2012)

Digilent PmodLS1 Infrared Light Detector Module Ref. Manual, Rev.A (2007)

IG-22GM DC Carbon-brush Geared Motor Series (2010)

Reflective Object Sensor OPB7xx Reference Manual, Issue B.3 (2009)

DeVry TWR Breadboard.pdf

ROBO_car_project.pdf (2013)

PmodLS1_sch.pdf (2007)

PmodHB5_sch.pdf (2006)

Appendix B Illustrations/Schematics

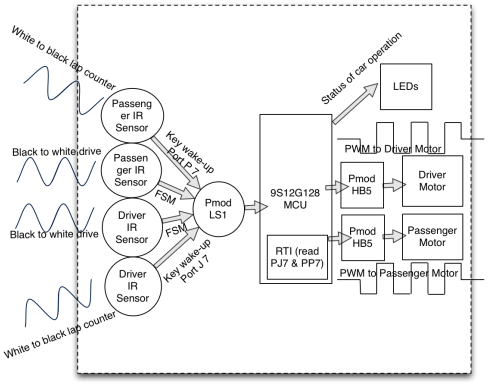


Figure 1: Hardware block diagram

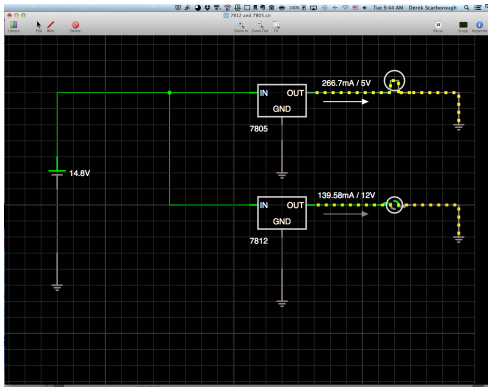


Figure 2: Power theory schematic

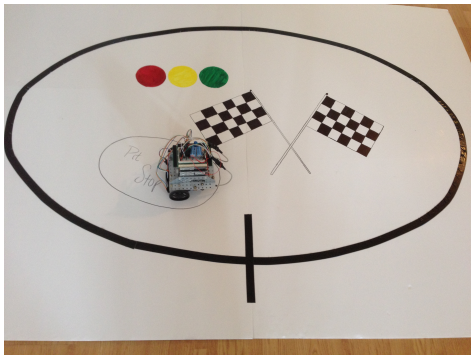


Figure 3: Robotic car track (80x60)

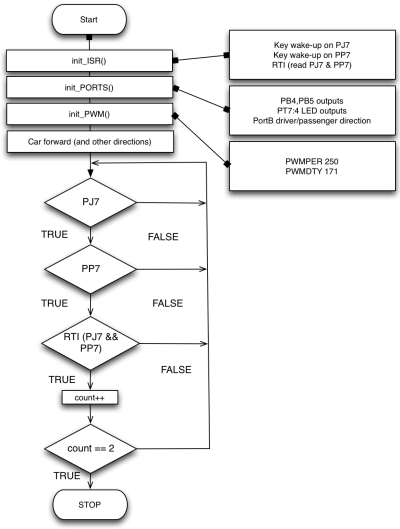


Figure 4: Flowchart

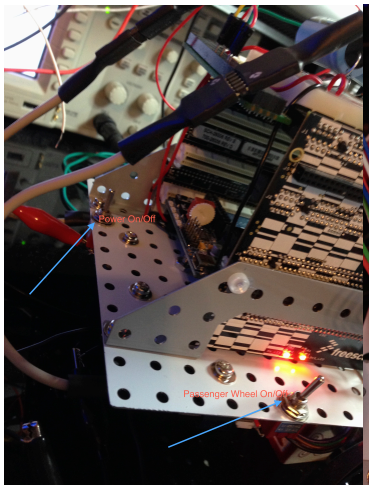


Figure 5a: Powering car

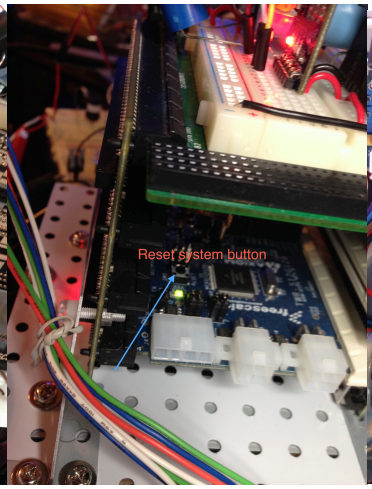


Figure 5b: Resetting car

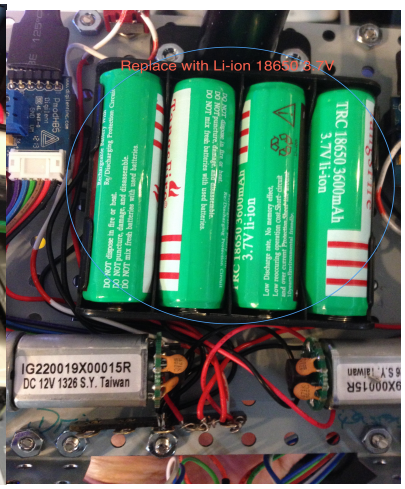
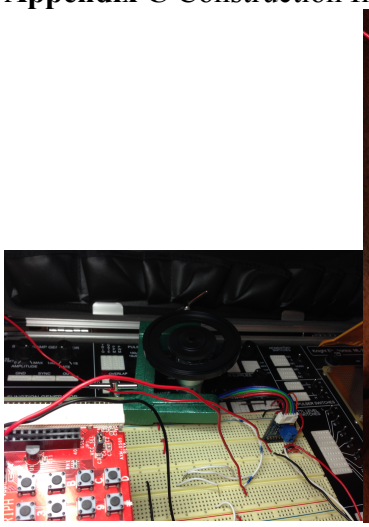
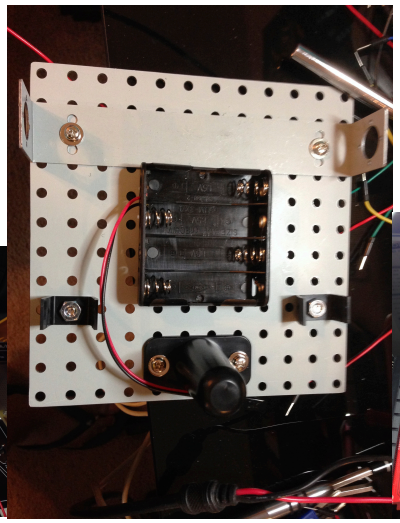
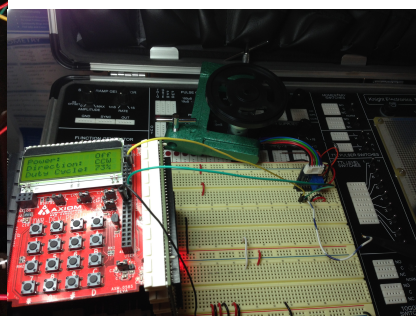
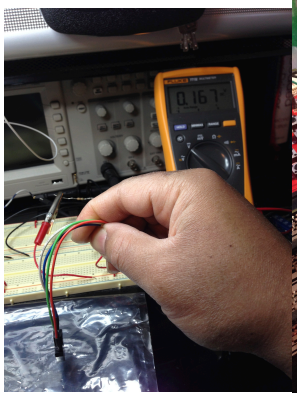
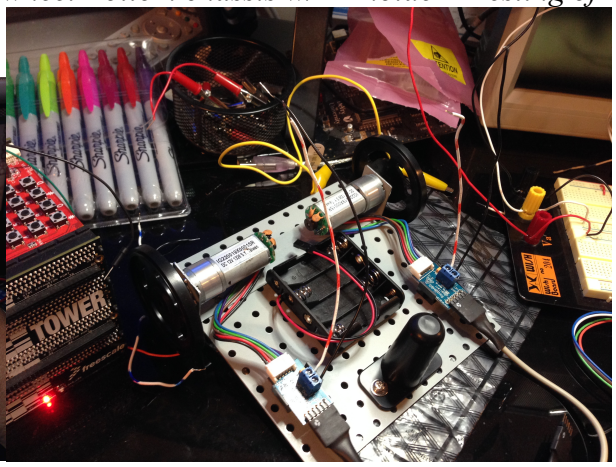


Figure 5c: Replacing batteries

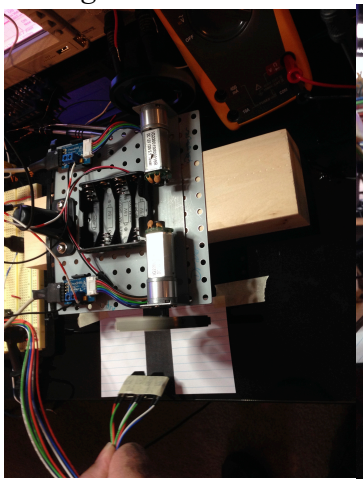
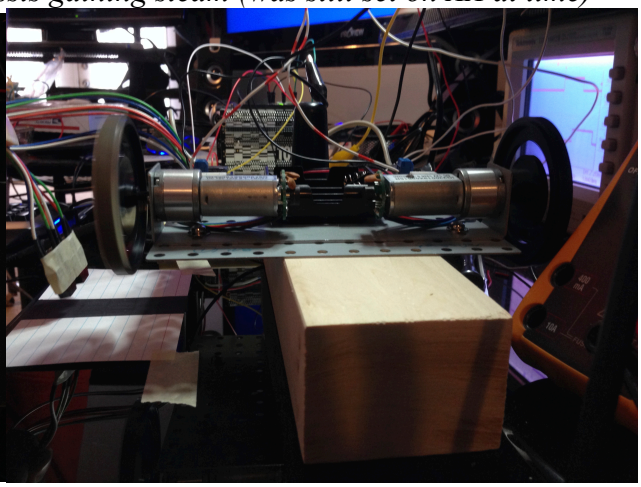
Appendix C Construction Illustrations

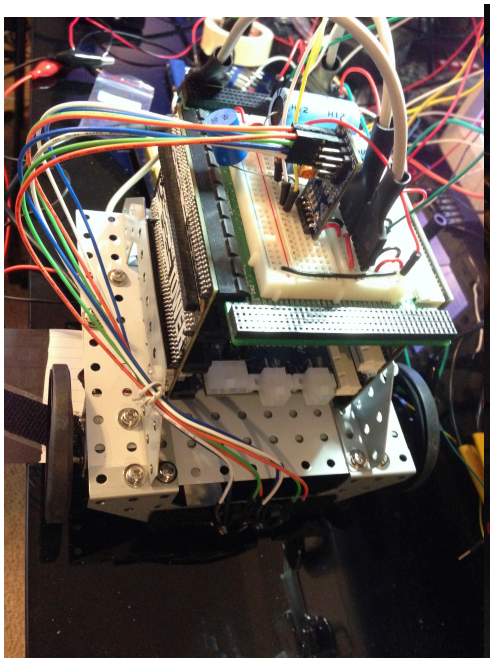
Humble beginnings-1 wheel Bottom chassis w/AA holder Testing of 1 motor and wheel

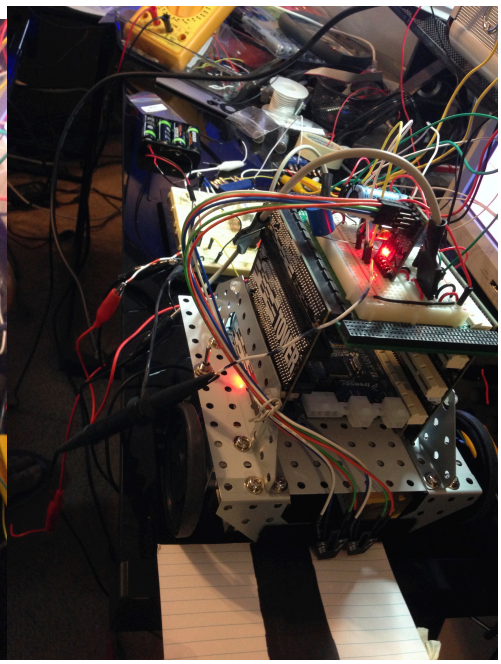
Testing IR sensor Chassis gaining steam (was still set on AA at time)

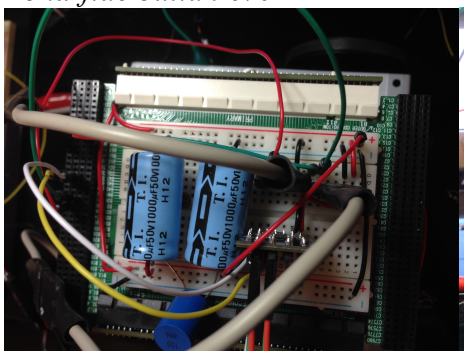
Testing IR sensors w/wheel More IR testing w/wheels



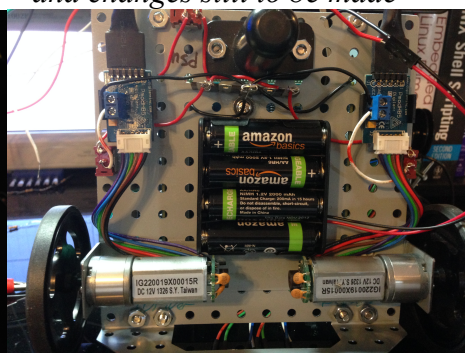
Bona-fide build 90%



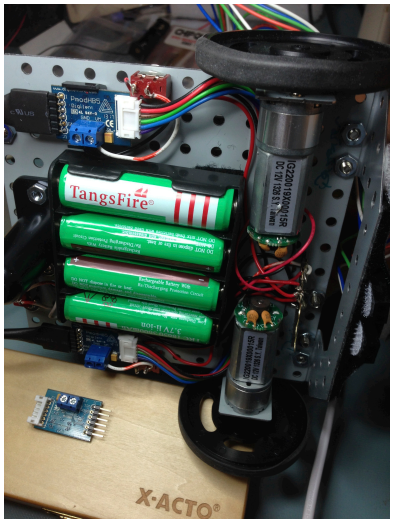
and changes still to be made



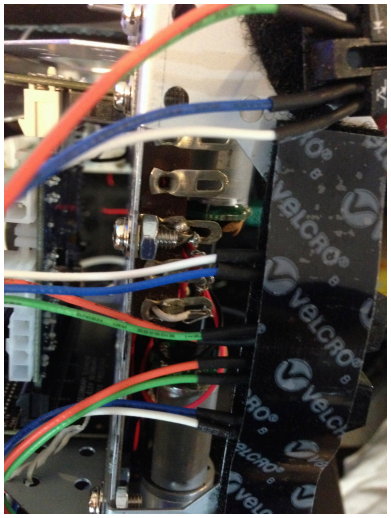
Breadboard full of goodies



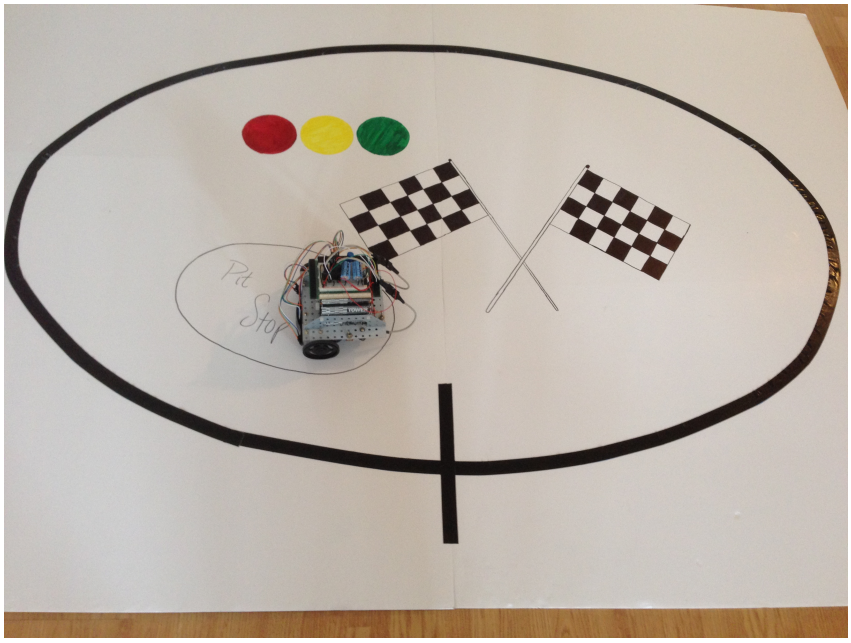
Still thinking of AAs?



Decision made-Li-ions for sure



Connect terminal with ground soldered



Finished product raring to go

REFERENCES

Hill, W. & Horowitz, P. (1999).*The art of electronics*. Cambridge [u.a.: Cambridge Univ. Press.

Mazidi, M. A., & Causey, D. (2009). *HCS12 microcontroller and embedded systems: Using Assembly and C with CodeWarrior*. Upper Saddle River: Pearson/Prentice Hall.

Monk, S. (n.d.). *Hacking electronics: An illustrated DIY guide for makers and hobbyists*.

Platt, C. (2012). *Encyclopedia of electronic components*.

Scherz, P., & Monk, S. (2013). *Practical electronics for inventors*. New York: McGraw-Hill.

Schildt, H. (2000). *C: The complete reference*. Berkeley: Osborne McGraw-Hill.

Valvano, J. W. (2012). *Embedded microcomputer systems: Real time interfacing*. S.l.: Cengage Learning.