Microcomputer-Controlled Car Project

Reference Manual
Version 1.0

University of Michigan - Tilbury Research Group

Last Updated April 20th, 2015

Table of Contents

Chapter 1: Overview

This is the microcomputer-controlled car project. This project's purpose is to make available an inexpensive test platform for ground vehicle control algorithms. Following are the major components of this project:

- 1 x RC racing car (AE Team Associated SC10)
- 1 x Raspberry Pi Model B microcomputer (revision 1 or 2)
- 1 x Arduino Pro Mini 5V 16MHz microcontroller
- 1 x Adadafruit 16 Channel 12-bit PWM/Serbo Driver – I2C Interface – PCA9685
- 1 x FTDI friend board from Adafruit
- Software for the Arduino written in the Arduino C-like language
- Python interface module for the Raspberry Pi to communicate with the Arduino
- Custom 5V power regulation circuitry

Chapter 2: RC Car

The RC racing car that was selected for this project is the AE Team Associated SC10, a 1:10 scale, ready-to-run 2WD, electric off road race truck. The stock car consists of the following major components:

- Independent suspension for the front and rear wheels
- 1 x Servo (XP DS1903MG) that steers the front wheels
- 1 x Radio Receiver 4 Channel 2.4GHz (XP TRS403-ssi)
- 1x Radio Transmitter (XP 2G 29215) 2.4GHz
- 1x Motor Controller (XP SC200)
- 1 x Ni-MH Battery Pack 7.2V 3600mAh
-

The motor controller must receive power from the 7.2V battery pack. The motor controller can and should be set to disable its low voltage cutoff when using a Ni-MH battery. The low voltage cutoff can be disabled by plugging in the motor controller to the battery, turning on the power switch, and pressing the SET button on the motor controller 2 times. The LED will flash green 5 times when low voltage cutoff is disabled. To turn on low voltage cutoff, press the SET button 2 times and the LED will flash red 5 times.

The servo, radio receiver, and motor controller are all compatible with 5V logic. The acceptable logic level range for the signals is between 4.8V and 6.0V. The PWM signals between the radio receiver and the motor controller and the PWM signals between the radio receiver and the steering servo operate at 72Hz. **NOTE: when reading the pulse width of the PWM signals with the Arduino pulseIn() function, the Arduino will report pulse widths that are not the true value, but will report them consistently.**

According to oscilloscope measurements, the steering servo's PWM input signal ranges from a minimum pulse width of 1.067ms to a maximum of 2.090ms. The Arduino's pulseIn() function, however, reports a minimum pulse width of 1.140ms to a maximum of 2.720ms. In both cases, a short pulse corresponds to a maximum right turn and a long pulse corresponds to a maximum left turn.

According to oscilloscope measurements, the motor controller's PWM input signal ranges from a minium pulse width of 1.020ms to a maximum of 2.020ms. The Arduino's pulseIn() function, however, reports a minimum pulse width of 1.304ms to a maximum of 2.650ms. In both cases, a short pulse corresponds to full reverse throttle and a long pulse corresponds to full forward throttle.

Chapter 3: Rapberry Pi (RPi)

The project currently uses a Raspberry Pi Model B revision 1. The Raspberry Pi Model B revision 2 is also compatible with a few minor tweaks to the RPi's configuration. Other models of the RPi were not tested with this project.

The RPi in this project runs the 2012-12-24 version of the raspbian-wheezy operating system. The following tutorial may be helpful for setup: http://elinux.org/RPi_Hub#Getting_Started

The RPi in this project uses Python and a package for python called pySerial. Python will most likely be pre-installed on the RPi's operating system but pySerial may need to be installed.

The RPi in this project is merely used as a platform to run a control algorithm for the car. The RPi is not strictly necessary for this project to function – the Arduino's software handles all of the signal reading and generation needed to control the car so a control algorithm could be run directly on the Arduino instead of on the RPi if it was written in the Arduino C-like language. The addition of the RPi, however, adds the ability to have the car controlled by a remote computer since the serial communication link between the RPi and the Arduino could be replaced by a wireless serial communication link such as XBee radios.

The RPi is connected to the Arduino via USB. One of the RPi's USB type A ports is connected to a mini-USB type B port on a FTDI breakout board which is connected to the Arduino. The RPi can send data to the Arduino using Python's serial module (pySerial). The pySerial interface was wrapped into a Python module called “car” which provides a standard interface of function calls for sending commands to the Arduino without having to construct the commands byte for byte. The basic idea is that the RPi will send a null-terminated string of characters that represents a command, to the Arduino.

The file “car.py” is the Python module that provides the API for the car. The API provides an class “Car” that represents the rc car. The three main class methods are setMode, setSteer, and setMotor. See the file “car.py” for complete documentation on the class.

Chapter 4: Arduino

The project currenty uses an Arduino Pro Mini 5V 16MHz microcontroller. The standard Arduino IDE was used for this project to develop the code.

The project uses some of the standard Arduino libraires (namely Serial) as well as a library written by Adafruit for the PWM board. The library for the PWM board must be downloaded and saved in the “libraries” sub-directory of the “arduino” project directory. The Adafruit PWM board library can be obtained at the following url: https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library.

The Arduino is connected to the Raspberry Pi via USB. The Arduino Pro Mini does not include FTDI circuitry so a FTDI board (available from Sparkfun or Adafruit) is connected to the Arduino's serial pins, then to the Raspberry Pi's USB port. The Raspberry Pi will send commands to the Arduino which controls the control mode, the steering angle, or the throttle of the car. The commands are sent as C strings which are then parsed and interpreted by the Arduino. Following is the format of each of the available comands (quotes are used only to show the beginning and end of the command strings:

- setMode: “<function> <mode><NULL>”
- setSteer: “<function> <direction> <percent><NULL>”
- setMotor: “<function> <direction> <percent><NULL>”

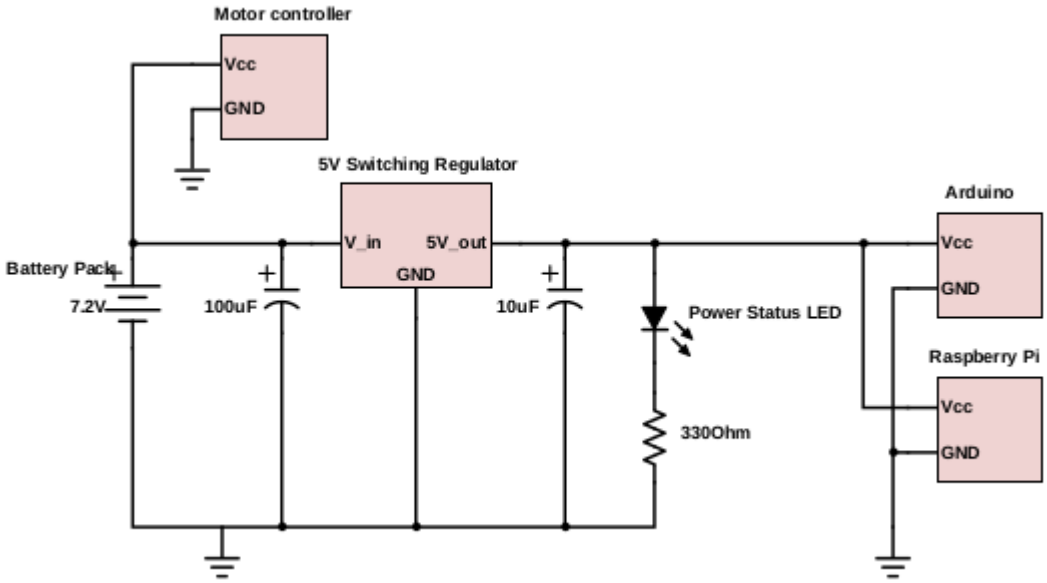
Following is the list of the function characters for each command:

- setMode: function = 'A'
- setSteer: function = 'B'
- setMotor: function = 'C'

The Arduino runs a program that continuously waits for a command and sets the PWM signals for the steering servo and the motor controller accordingly. The program has 3 control modes, “idle” (do nothing), “rc” (controlled by the radio transmitter), and “rpi” (controlled by the serial commands sent by the Raspberry Pi). The Arduino will always respond to a serial “setMode” command, however, regardless of the mode it is currently in.

Chapter 5: Power Circuitry

The project uses custom power regulation circuitry to provide a steady 5V DC to the Raspberry Pi and the Arduino. The 5V switching regulator used is the UBEC DC/DC Step-Down (Buck) Converter - 5V @ 3A output, available from Adafruit. Below is a circuit diagram of the power circuitry.



Chapter 6: Future Plans

Plans for improvement in the next versions of the project include:

- Improve serial command structure
- Integrate IMU
- Integrate robotic arm and accompanying step-up power circuitry
- Add encoder to rear wheel axle