Joel Gomez
861238902
CS120B-021
June 5, 2016

CS120B Spring 2016 Custom Project - AR/CI

Overview

Project Automatic and Reactive Collision Intervention (AR/CI ("arr-see")) was set out to be a remote-controlled vehicle with onboard collision avoidance and operator assistance features that allow for semi-autonomous operation. By design, the vehicle is intended to be "driven" by an operator through a remote device for recreational fun, but was to be equipped with hardware and logic to prevent damage to the vehicle itself that may come as a result of operator negligence or accidental circumstances.

Components

Vehicle - A "donor" chassis (axles, steering components, gears, etc.) from an existing radio-controlled car is used
Movement - a DC motor is used to actuate rotational motion for the purpose of propelling the vehicle forward/backward
Steering - a servo is used for operating the steering mechanisms to establish vehicle direction
Collision Prevention* - distance sensors will be used to provide the vehicle a sense of what's in front and around it
Control - remote control is done wirelessly via encrypted Wi-Fi connection. A Raspberry Pi Zero provides a secure access point (AP) and a web server to host the control interface to connecting client
Controller - the Atmega1284 is the primary microcontroller and gateway between all of the components
Human Machine Interaction - a smartphone or any device with a web browser (PC, tablet, etc.) is used to provide user input/commands to operate the vehicle.

**Collision Prevention was not implemented in final design*

Operation

AR/CI is controlled via a virtual joystick[1] that is drawn onto a webpage hosted on AR/CI's onboard WiFi Access Point. To command AR/CI, the user must search for and connect to the ar-ciAP access point with the secure passphrase. Once connected, AR/CI's onboard DNS server routes ANY traffic through it's localhost, so the user just needs to attempt to visit any website and he/she will automatically be redirected to AR/CI's control interface. Alternatively the user may point his/her browser to the AR/CI Gateway I.P. 10.0.0.1.

The user interface is both touchscreen and mouse-friendly - a virtual joystick is drawn to the screen upon touching or clicking anywhere on the window canvas. AR/CI moves relative to the direction the joystick is pulled once drawn.

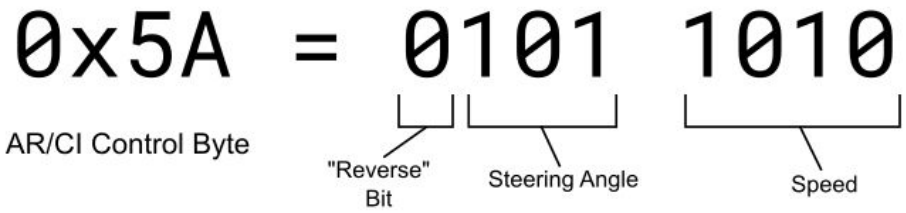
1 - virtual joystick by jeromeetienne, https://github.com/jeromeetienne/virtualjoystick.js

Behind the Scenes

While AR/CI's construction may seem trivial from afar, there is actually a lot going on "under the hood". We'll start at the ATmega1284.

ATmega1284

The ATmega1284 manages AR/CI's powertrain. Communication to the ATmega1284 is done via USART on channel 0. The ATmega1284 expects a byte of data containing the speed and steering angle for AR/CI. The most significant nibble is split up to determine direction (forward or reverse) on bit 7, and a steering angle position between 0-6 on bits 6, 5, 4. The least significant nibble determines the speed or duty cycle to drive the DC motor within a range of 0-15.



Steering

AR/CI's original donor chassis used a secondary DC motor to steer the frame either fully left, or fully right. For more precise control, the DC motor was discarded and a servo was fitted onto the steering linkage in order to to be allow the user to steer AR/CI fully left, fully right, or anywhere in between.

The steering angle is read on the bits 6, 4, 5 of a control byte where a value of 0x0- is fully right a value of 0x6- is fully left, and 0x3- is center. The steering angle is converted to a PWM duty cycle that is sent to the servo at 50Hz using the ATmega1284's Fast PWM feature with Timer3 and the OC3A pin.

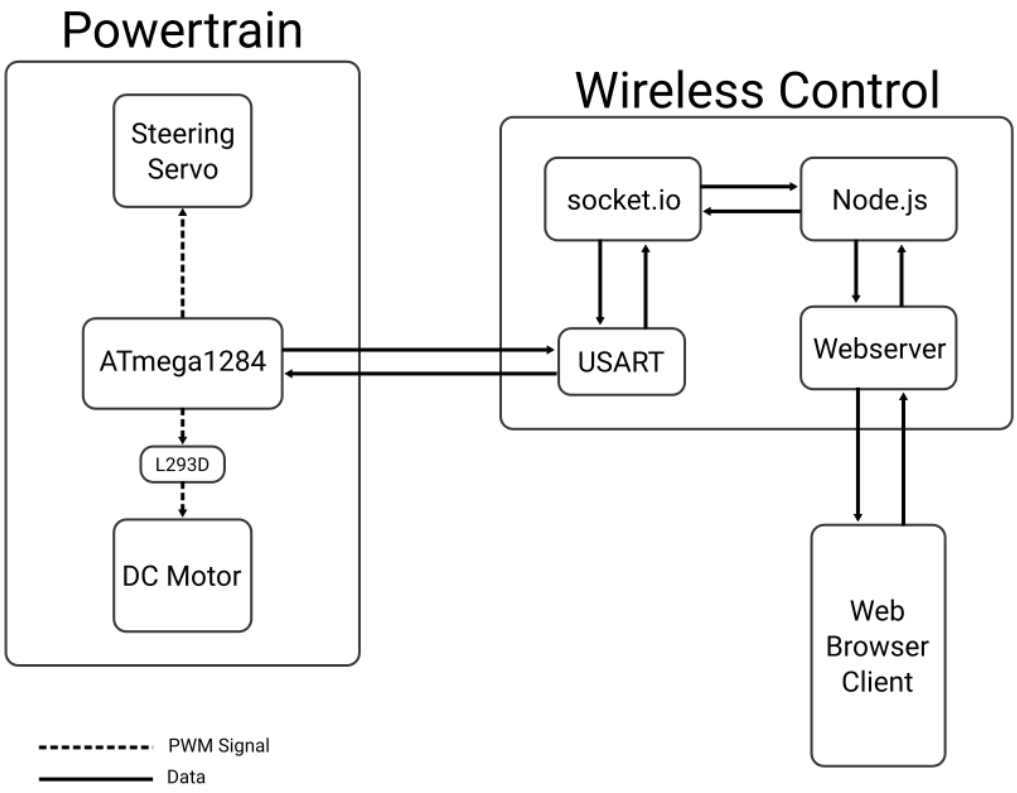
Forward/Reverse

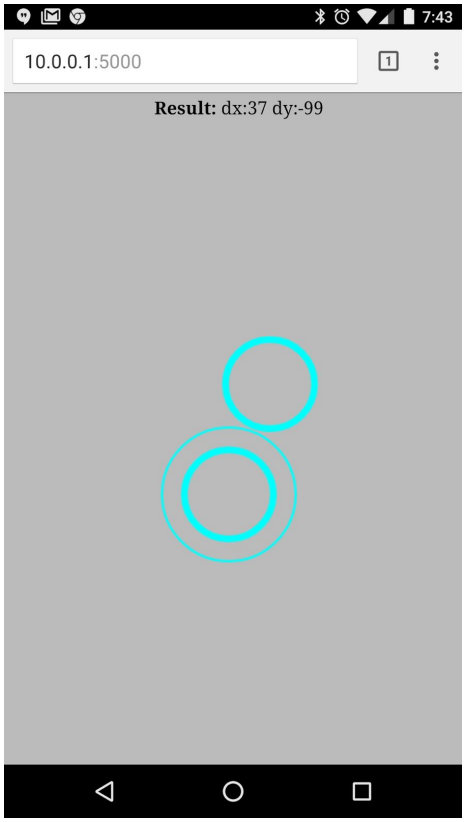
The speed for forward or reverse direction is read on bits 3, 2, 1, 0 of a control byte and ranges from 0 (no movement) to 15 (full speed). The direction of the motor is determined by bit 7 of a control byte, where a request for reverse movement is done by setting bit set 7. The speed is converted to a PWM duty cycle that is sent to the L293D motor controller at 250Hz using the ATmega's Fast PWM feature with Timer1 and pins OC1B and OC1A, depending on the direction.

User Control

The wireless access point "ar-ciAP" is broadcast and served using a Raspberry Pi Zero with the stock Raspbian Lite image. A DNS server ensures that any HTTP request from a connected

client (user) is routed to AR/CI's control interface, hosted on port 5000. A webserver and Node.js host a static web page that runs some client-end javascript that draws the virtual joystick onto the screen and sends the position data back to to the Node.js script via websockets with socket.io. The Node.js script then routes the user input out through the Raspberry Pi's USART channel at baud 9600.





AR/CI mobile control interface shown on Android phone.

Testing

AR/CI was tested at various stages in development starting with simple “bench” testing to ensure proper control via control bytes sent manually over a PC serial port to verify basic functions. From there, the control interface was tested and debugged in software over a virtual machine that would fill in for the Raspberry Pi during development until it was ready to test on the Pi itself. Once the WiFi access point seemed to be working, it was tested for correct functionality by writing the data to a serial port on another PC and verifying the output with a serial console on a PC.

The first prototype was tested on a platform to prevent the chassis from physically moving and running amuck in the event of control failure.

Through testing, the need for a timeout controller was found in the event that any point in the communication circuit failed while AR/CI was moving. Communication was frequently interrupted during the first trials and AR/CI would end up going rogue and running away until it crashed. The communication interrupts were found through PC debugging and determined to be coming from the serial port on the Raspberry Pi crashing after a random amount of data flow - this was resolved in software by adding additional checks on the Node.js script to verify that the serial port was open before attempting to write to it.

Secondary testing was handed off to random users that were not part of the project or engineering students to ensure user friendly-ness of AR/CI control. After resolving some latency issues with a faster system period, all tests seem to pass and AR/CI was determined ready-enough for regular use.

Known Bugs

Once in awhile, serial communication between the ATmega1284 and the Raspberry Pi is interrupted, it's less than 10% of the time, but it *does* occur. This is could possibly be related to network latency issues, the serial port on the Raspberry Pi crashing again, or the Node.js process being killed by the Linux kernel on the Raspbian OS in an attempt to free up resources or recover from a fault. Due to time constraints and the relatively low frequency of the issue, the exact cause and resolution has not been found as of this report.

Shortcomings

AR/CI was originally supposed to have onboard crash detection and directional orientation. Due to time constraints, crash detection was coded in and tested, not implemented in the final design. A separate microcontroller must manage the sonar sensors for crash detection in order to preserve multi tasking and latency requirements of AR/CI's powertrain control. AR/CI's completed source code monitors PA0 and PA1 for an external signal indicating that a obstacle is detected in front of or behind AR/CI respectively and prevents movement in the appropriate direction until cleared while still allowing movement in the opposite direction for course correction.

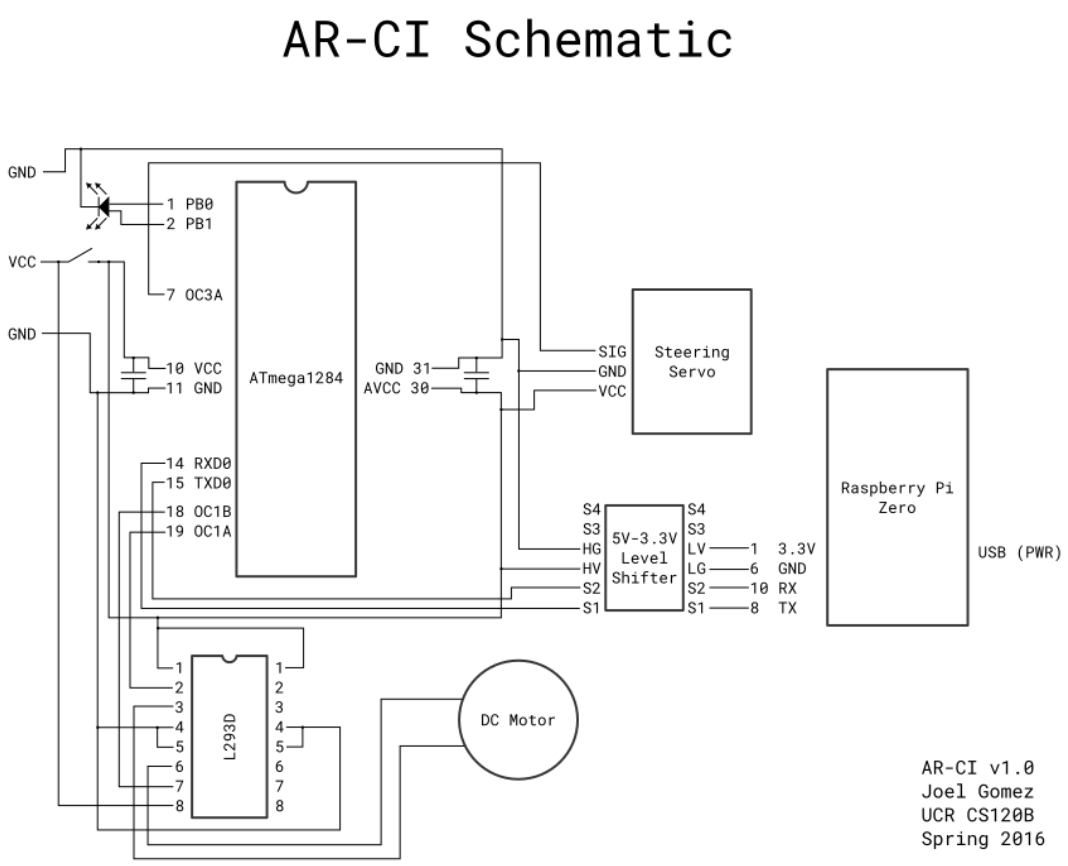
Directional orientation was abandoned after further thought and realization that it would not work as intended. Originally, it was planned for the user to control AR/CI by moving the joystick in whatever direction he/she wanted AR/CI to go. This relied not only on AR/CI knowing the direction of heading, but also AR/CI's current position relative to the user. This feature was determined to be too small of a benefit for the amount of work it would require.

Appendix

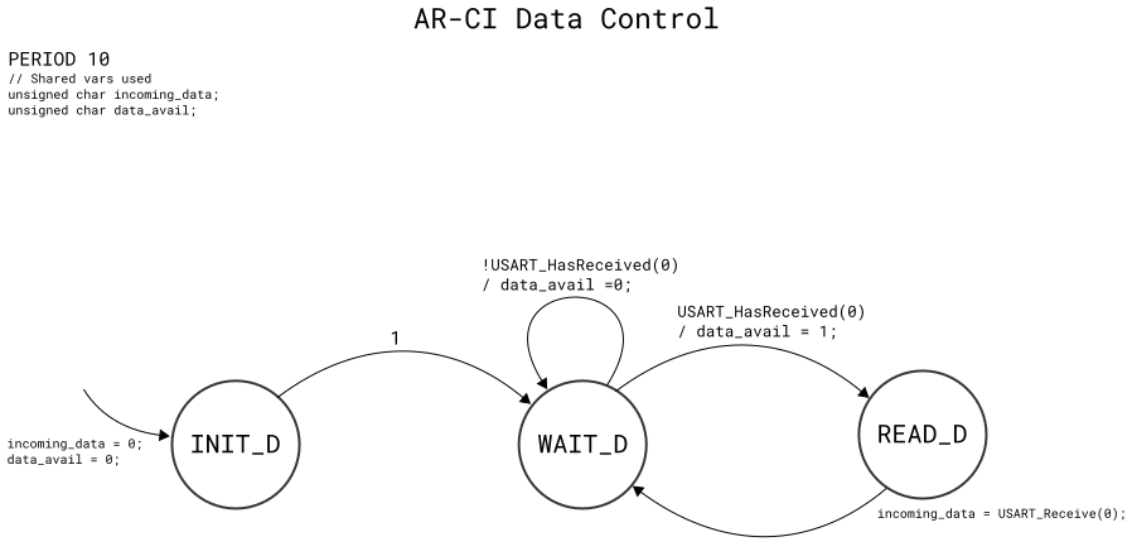
Resources

Project website - http://joelgomez.me/arci
Project github - https://github.com/jgome043/arci/
Project video - https://youtu.be/WWuufRkIJ9c

Schematic


AR-CI Schematic
GND
VCC
GND
1 PB0
2 PB1
7 OC3A
10 VCC
11 GND
ATmega1284
GND 31
AVCC 30
SIG
GND
VCC
Steering Servo
14 RXD0
15 TXD0
18 OC1B
19 OC1A
S4
S3
HG
HV
S2
S1
5V-3.3V Level Shifter
LV
LG
S2
S1
1 3.3V
6 GND
10 RX
8 TX
Raspberry Pi Zero
USB (PWR)
L293D
DC Motor
AR-CI v1.0
Joel Gomez
UCR CS120B
Spring 2016

State Machines

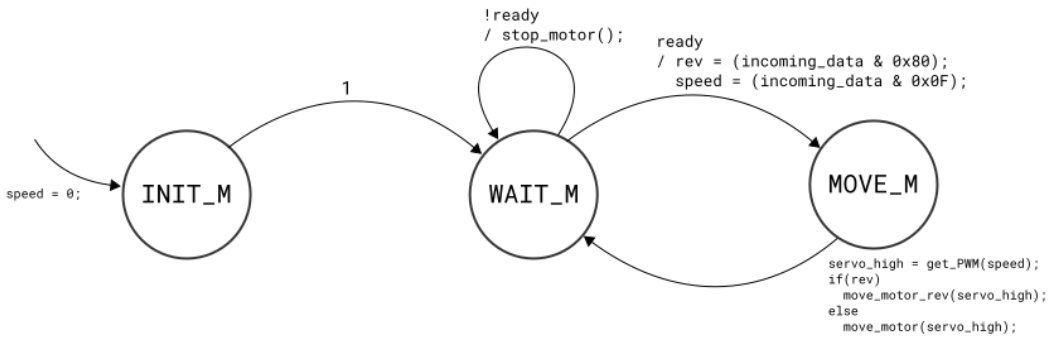


AR-CI Motor Control

PERIOD 10

```
// Shared vars used
unsigned char incoming_data;
unsigned char ready;

// Local vars
static unsigned short motor_high;
static unsigned char speed;
static unsigned char rev;
```

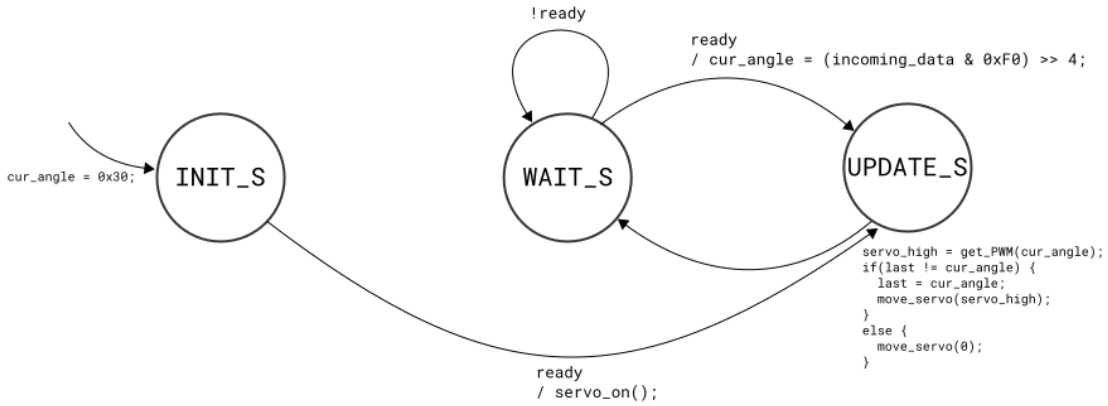


AR-CI Steering Control

PERIOD 10

```
// Shared vars used
unsigned char incoming_data;
unsigned char ready;

// Local vars
static unsigned short servo_high;
static unsigned char cur_angle;
static unsigned char last;
```



AR-CI Timeout Control

PERIOD 10

```
// Shared vars used
unsigned char incoming_data;
unsigned char ready;
unsigned char data_avail;

// Local vars
static unsigned char st_led;
static unsinged char elapsed;
static unsigned char ready_cnt;
```