## Active Steering and Torque Vectoring for Electric Vehicle

## ECE 4220 – Real Time Embedded Computing

## Project Report

## Naasa Fikri

## University of Missouri – Columbia

## May 15, 2015

**Abstract**

In this project, we are going to demonstrate the implementation of active steering and torque vectoring in an electric vehicle. We a using two small servos to emulate the work of the steering rack and the brake. We are also using 2 small electric motors to emulate the work of two propulsion motors. For the input, we are using an eight channel ADC to read 4 potentiometer values to emulate steering angle sensor, brake sensor, differential bias, and accelerator pedal.  
**Introduction**

In recent years, there is a shift in people’s attitude towards energy. In the transportation sector, people has mostly been using fossil fuel. The trend has changed because fossil fuel is polluting and also depleting in quantity. Nowadays, people tend to consider hybrid of fully electric vehicle. Not just an ordinary electric vehicle, but the one that is efficient and has a really good performance (acceleration, handling, etc.).

In this project, we want to emulate the work of active steering and torque vectoring. This project is quite useful to be implemented in electric or hybrid vehicle because it can improve both the performance and efficiency of the vehicle. The objectives are:

1. Control the steering servo from a steering wheel angle sensor
2. Control the brake actuator servo from the brake sensor
3. Implement the torque vectoring depending on the steering input. I.e. send different torque to the left and right motor.
4. Implement differential open/lock so that we can have two wheels running on different speed (open) or at the same speed (lock).

As mentioned in the abstract, we have 4 potentiometers to emulate steering wheel angle sensor, brake actuator sensor, differential bias, and accelerator pedal. We have one servo working as a steering rack and another servo working as a brake actuator. We also have two electric motors each for the left and right wheel. The torque vectoring works by varying the power to the driving left and right wheel. The resultant of the two differences will point to the direction of travel of the vehicle.

**Background**

In this project, we found a difficulty to find the exact algorithm to determine the torque needed to be sent to each wheel when the vehicle is turning. Thus, we have developed our own algorithm to achieve the torque vectoring.

This technology is implemented in many high performance vehicles using mechanical approach. The torque is divided to the left and right wheel using a differential connected to a clutch. In electric vehicle however, we do not want to waste energy in the clutch and the differential. Some example of student built vehicle using torque vectoring:



Figure 1. Delft University's DUT 15. Utilizing 4 electric motors in each wheel to implement torque vectoring. Champion of Formula Student 2014.



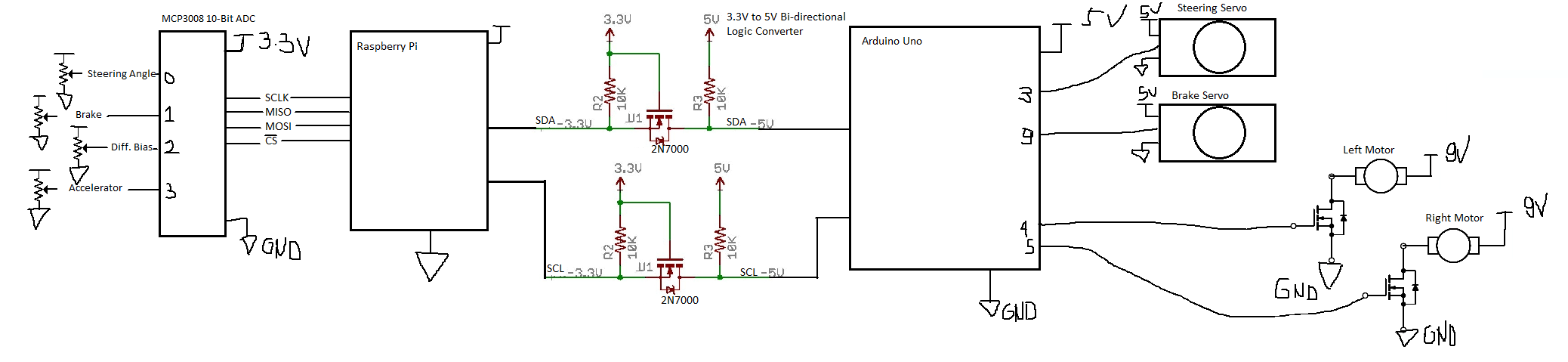
Figure 2. ETH Zurich's "grimsel". Using 4 electric motors on each wheel to do torque vectoring. Current Guinness World Record holder for the fastest acceleration for electric vehicle. 0-100km/h in 1.785 seconds

As we can see, both vehicles are using one electric motor for each wheel. However, we do not know how they implement the torque vectoring.

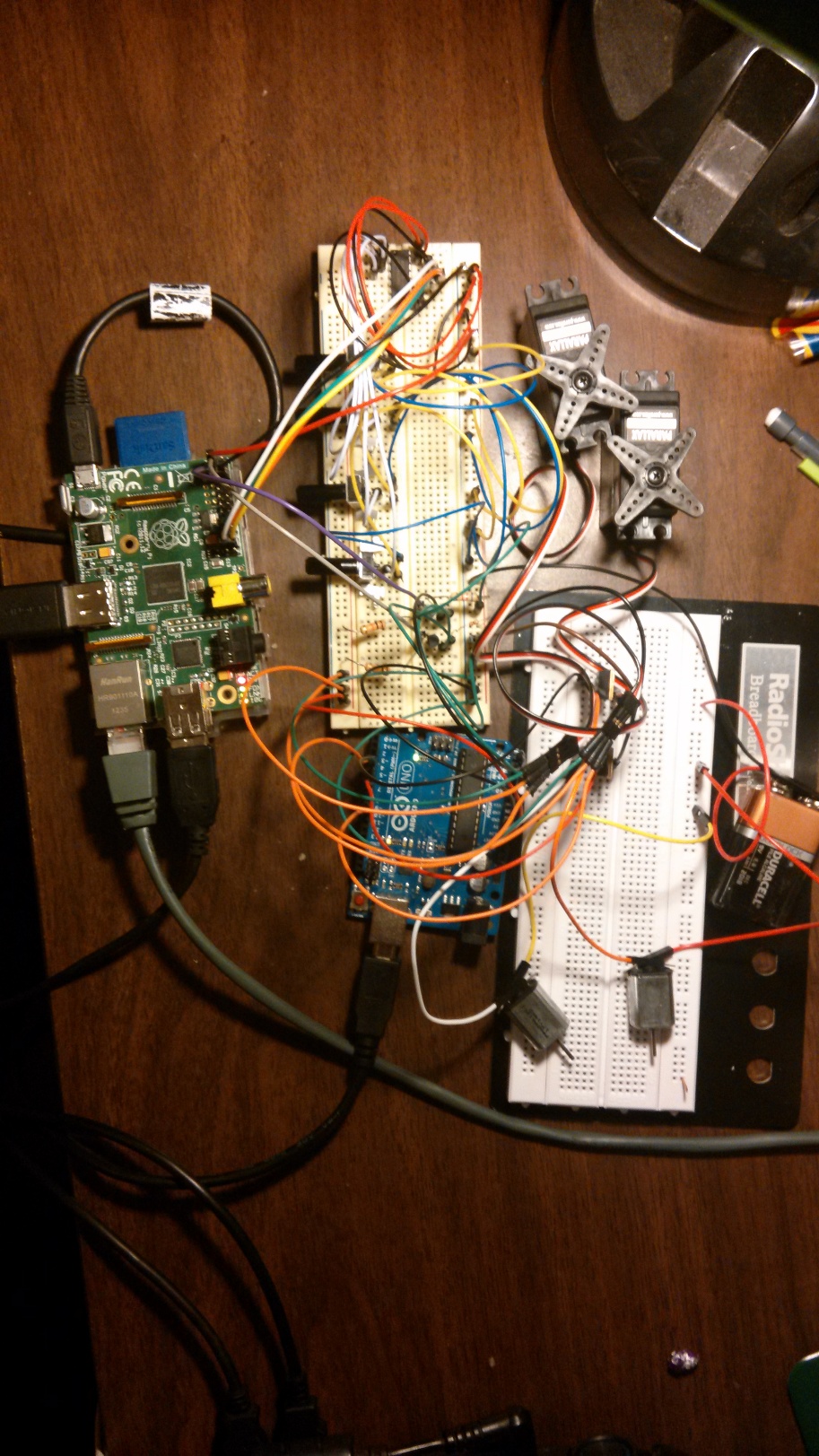
**Implementation**

In this project, we are using a Raspberry Pi as the main computer. We are using a Raspberry Pi because it has 4 CPU cores so that we can use multiple threads. It also runs on Linux so that we can program it in C. This Raspberry Pi is communicating with and an ADC through SPI. The ADC reads 4 potentiometer inputs. The Raspberry Pi will process the data read from the ADC and then send it to an Arduino through I2C. The Arduino will control all of the servos and motors. The Arduino is ideal for the project because it has no operating system so that we are sure about its real time capability. It is also programmed in C-like language.

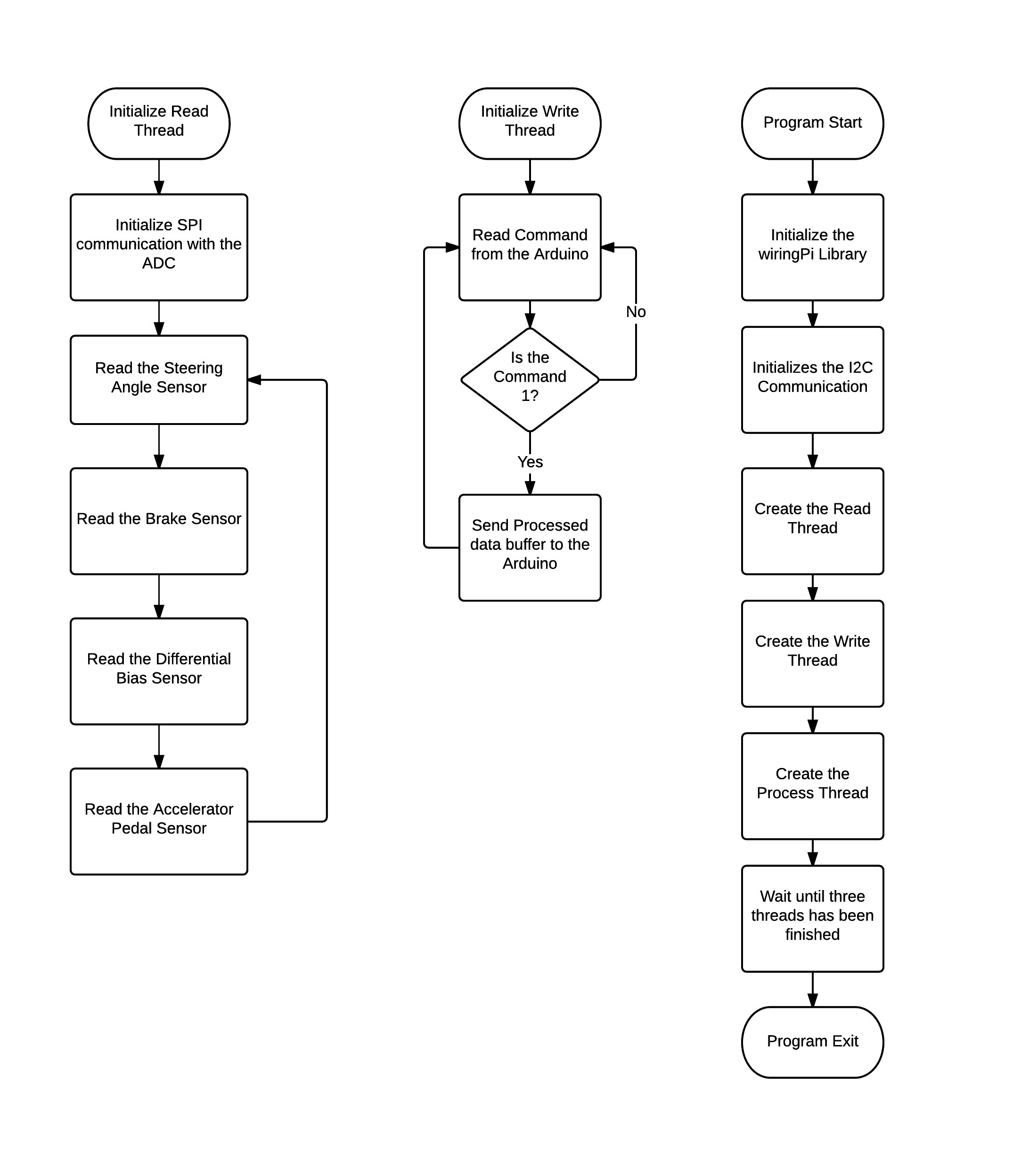
Here is the Hardware schematic:

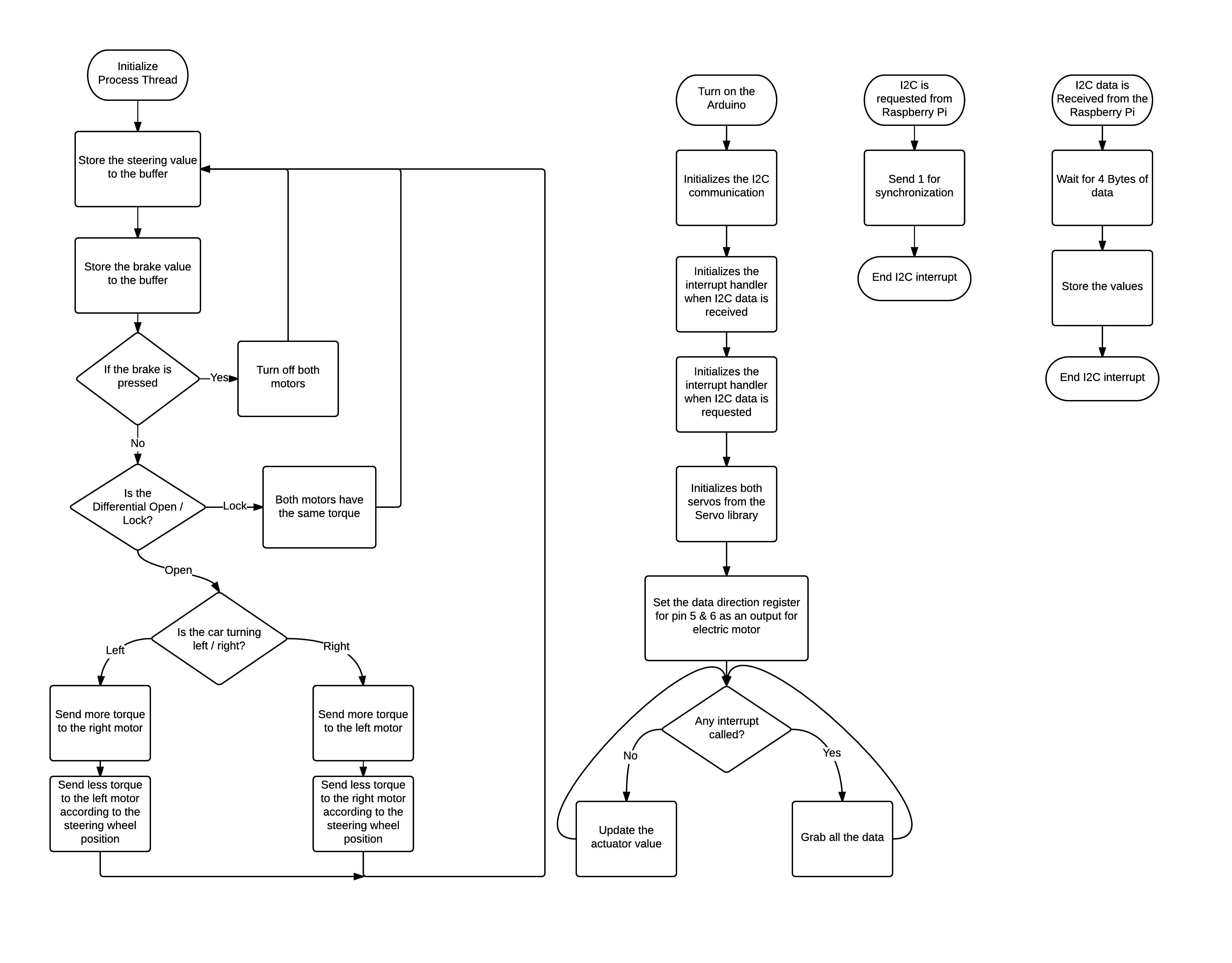


Circuit

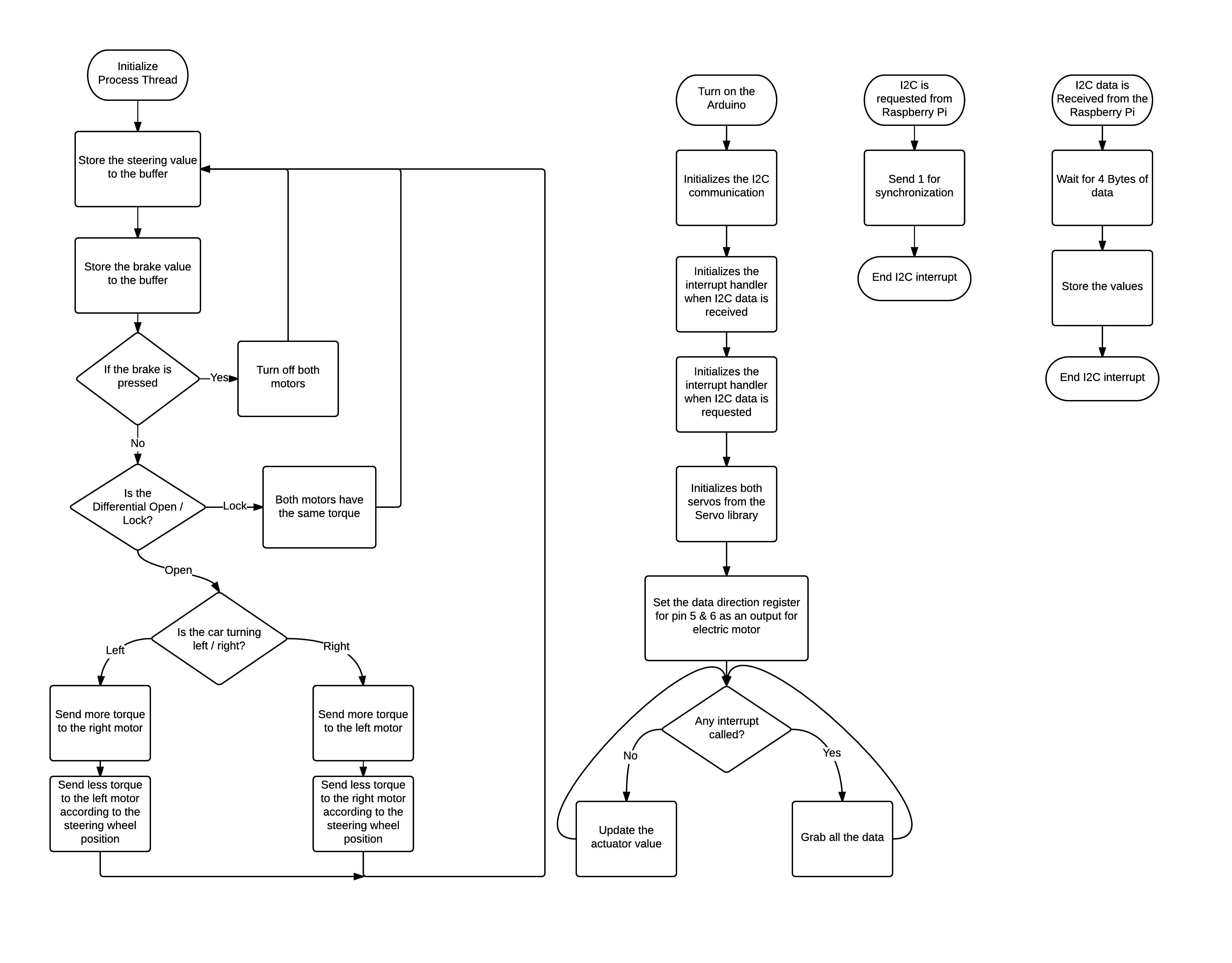


Here is the Flowchart in the Raspberry Pi Program:





Arduino Flowchart:



The main program

First of all, we initialize the wiringPi library that is going to handle the SPI and I2C communications. We then call the I2C initializer that will return the file descriptor for I2C communication. Then, we initialize all of the threads that will handle the sensor reading, sensor data processing, and sensor that will send the processed value to the Arduino.

Sensor reading thread

The thread will initialize the SPI communication with the ADC. The Raspberry Pi then read all of the potentiometer values and store it in 4 variables in an infinite while loop.

Sensor data processing thread

This thread will read the value from the variables and decide what the actuator should do. The processed data is stored inside a 4 bytes array. The zeroth byte is the steering rack angle value. We divide this from 1024 value (10-bit) to 170 degrees steering angle by dividing it by 6. We need to do so because the Arduino servo takes the degree of rotation as the servo input. We also do the same thing for the brake servo, which is the first byte of the array.

The thread then determines the amount of torque sent to the left and right wheel. First of all, it is checking the brake. If the brake is pressed, it will set the second byte (left motor torque) and the third byte (right motor torque) to zero. The Arduino takes 8- bit of data so that we have to divide the 10-bit data from the ADC by 4. Then, if the brake is not pressed, the program checks the differential bias value. If the differential is set to lock, both motors will have the same torque. Thus, we set the second and the third byte to whatever the accelerator pedal value is, divided by 4.

If the differential is open, the program will check the steering angle. If the car is turning left, it will set the third byte (right motor) to the accelerator value and set the second byte (left motor) to the multiplication of the accelerator value and the steering angle. The straighter the vehicle, the more torque is sent. When the car is turned all the way to the left, the steering value will be zero and zero torque will be sent to the left motor. Everything in between is linear.

If the car is turning right, the program will send the accelerator value to the left motor. However, we need to invert the multiplier of the steering wheel. We are doing this by subtracting 1023 (maximum ADC value) with the steering angle. If the vehicle is straight, it will read around 512. If the vehicle is turned all the way to the right, it will read zero. Thus, it will send zero torque to the right motor. Everything in between is also linear.

Data writing thread

The buffer that has been processed by the processing thread is now ready to be sent to the Arduino. The thread is waiting until the Arduino is ready by signaling 1 to the Raspberry Pi. When the Arduino is ready, the thread will send 4 bytes buffer to the Arduino through I2C.

Arduino setup

Initially, the Arduino will start the I2C communication in address 0x04. Then, it will register the interrupt handler either when I2C data is received (Raspberry Pi write system call) or I2C data is requested (Raspberry Pi read system call). It also initializes the servos and electric motor to the respective pins. When the data is requested, it will send the value 1 to the Raspberry Pi. When the data is received, it will wait until 4 bytes of data has been collected. The zeroth byte will be sent to the steering servo. The first byte is assigned to the brake servo and the second and third byte will be sent to the left and right motor respectively. Inside the main block, the Arduino assign the value to the servos and the electric motor using Pulse Width Modulation.

**Experiments and Results**

First of all, we tested the communication between both devices to make sure that the bytes sent are in sync. The Raspberry is checking it through the printf() function while the Arduino is using Serial.println() function. Then, we connect all of the servos and motors. First we test the steering servo. When we turn the steering potentiometer, the steering servo is also turning. There is a little delay between them but it is linear with the potentiometer. The delay is necessary for the Arduino so that it will not freeze because it is running out of resources processing the I2C communication. We did the same thing on the brake servo and the same result repeats.

We then test the dynamic aspect of the system. We set the brake to zero and lock the differential. When we increase the accelerator value, both motors start turning. The speed is linear to the accelerator value. The motor speed was not changing even when we change the steering angle. When we turn the brake potentiometer, both motors stop turning.

We then open the differential and set the brake to zero. We put the steering wheel all the way to the left and increase the accelerator pedal. The right motor is spinning but the left motor did not turn at all. When we decrease the steering angle by turning the steering wheel towards the center, the left motor start turning. The left motor is also responsive to the steering wheel angle. When we increase the steering angle towards right, the right motor started to slow down until it fully stopped when the steering wheel is all the way to the right.

**Discussions and Conclusions**

When we tested the system using only Raspberry Pi, the servo motor is jittery and having a non-deterministic behavior. Thus, the system is not suitable for real time operation. This might be because of the Operating System’s resource allocation. We then decided to use an Arduino to handle all of its actuations. We realized that the Arduino is running on 5V while the Raspberry Pi is running on 3.3V. We need to put the Bidirectional logic converter circuit because we are using I2C, which is a bidirectional communication.

We then had a problem with the bytes not in sync with the Raspberry Pi. The Arduino prints garbage values and the data size is not always 4 bytes. Thus, we cannot parse the data to be sent to the actuators. We then implement the data syncing with between the Arduino and the Raspberry Pi. When the Arduino is ready, it will send the value 1 to the Raspberry Pi. Then, the Raspberry Pi send 4 bytes of data to the Arduino. Every time the interrupt is called, the buffer will be flushed that is why the communication is in sync. There are no garbage data from the previous transaction.

Everything was well thought before the circuit was connected. This is quite a complex system because every part of the input will change the output. The steering and brake servo is linear to the potentiometer input. Both motors are also responding to the steering input, brake input, differential bias input, and the accelerator input. Both the servos and electric motors are also responding to the change in input simultaneously. In conclusion, the project objective has been accomplished.

This implementation can be easily transferred to road vehicle. We just need to use larger actuator that can actually handle the steering forces and the vehicle propulsion power needed by the vehicle. We can also implement a closed-loop control of the motors. The system constantly checks the motor speed and adjusts the speed delivered to each motor accordingly. We can also use four propulsion motors instead of two motors so that we can have higher traction and more precise torque vectoring.

**Appendices**

Raspberry Pi Code:

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

#include <wiringPi.h>

#include <wiringPiSPI.h>

#include <wiringPiI2C.h>

#include "mcp3004.h"

#define BASE 100

#define SPI\_CHAN 0

int i2cHandler; //file descriptor for i2c communication

int steeringVal, brakeVal, biasVal, accelVal; //value read from the ADC

char buffer[4]; //store the processed value

char inputBuffer; //i2c input buffer. receive command from the Arduino

void readVal(){

mcp3004Setup (BASE, SPI\_CHAN) ; //ADC initialization setup from the library

while(1){

steeringVal = analogRead(BASE); //read values from ADC

brakeVal = analogRead(BASE+1);

biasVal = analogRead(BASE+2);

accelVal = analogRead(BASE+3);

printf("%d %d %d %d\n", buffer[0], buffer[1], buffer[2], buffer[3]); //print the processed //value

}

}

void writeVal(){

while(1){

read(i2cHandler, &inputBuffer, 4); //read command from the Arduino

if(inputBuffer == 1){

printf("%d\n", write(i2cHandler, &buffer, 4)); //write the processed value to the

//i2c bus

}

else{

}

usleep(1000); //delay

}

}

void processVal(){

while(1){

buffer[0] = steeringVal/6; //divide the steering value from 0-1023 to 0-170 degrees

buffer[1] = brakeVal/6; //divide the brake value from 0-1023 to 0-170 degrees

if(brakeVal > 10){ //if the brake is pressed, the motor will be turned off

buffer[2] = 0;

buffer[3] = 0;

}

else{

if(biasVal < 512){ //if the differential is opened

if(steeringVal < 512){ //turning left

buffer[2] = (steeringVal \* accelVal / 512) / 4; // right motor

//speed will depend on the steering angle

buffer[3] = accelVal / 4;

}

else{ //turning right

buffer[2] = accelVal / 4;

buffer[3] = ((1023 - steeringVal) \* accelVal / 512) / 4;

// left motor speed will depend on the steering angle

}

}

else{ // differential is fully locked. both motors run at the same speed

buffer[2] = accelVal / 4;

buffer[3] = accelVal / 4;

}

//accelerator is divided by 4 to adjust the value from 0-1023 to 0-255

//because the data is sent by byte and Arduino DAC only handle 8 bit of data

}

}

}

int main(){

wiringPiSetup(); //initialize the wiringPi library

i2cHandler = wiringPiI2CSetup(0x04); //initialize the i2c communication.

printf("%d\n", i2cHandler);

pthread\_t readThread, processThread, writeThread; //pthread initialization

pthread\_create (&readThread, NULL, (void \*) &readVal, NULL);

pthread\_create (&writeThread, NULL, (void \*) &writeVal, NULL);

pthread\_create (&processThread, NULL, (void \*) &processVal, NULL);

pthread\_join(readThread, NULL);

pthread\_join(writeThread, NULL);

pthread\_join(processThread, NULL);

return 0;

}

Arduino Code:

int steeringVal, brakeVal, leftMotor, rightMotor; //buffer value

#include <Wire.h>

#include <Servo.h>

Servo steeringServo, brakeServo;

void setup()

{

Wire.begin(4); // join i2c bus with address #4

Wire.onReceive(receiveEvent); // initialize interrupt when data is received.

Wire.onRequest(requestEvent); // initialize interrupt when data is requested.

Serial.begin(9600); // start serial for output

steeringServo.attach(3); //initialize the servo

brakeServo.attach(9);

pinMode(5, OUTPUT); //initialize both motors. set the data direction registers, etc.

pinMode(6, OUTPUT);

}

void loop()

{

delay(100);

steeringServo.write(steeringVal); //write the values

brakeServo.write(brakeVal);

analogWrite(5, leftMotor);

analogWrite(6, rightMotor);

}

// function that executes whenever data is received from master

// this function is registered as an event, see setup()

void receiveEvent(int howMany)

{

while (4 < Wire.available()) // loop through all but the last. Never gets executed

{

}

steeringVal = Wire.read(); // receive byte as an integer

Serial.println(steeringVal); // print the integer

brakeVal = Wire.read();

Serial.println(brakeVal);

leftMotor = Wire.read();

Serial.println(leftMotor);

rightMotor = Wire.read();

Serial.println(rightMotor);

}

void requestEvent()

{

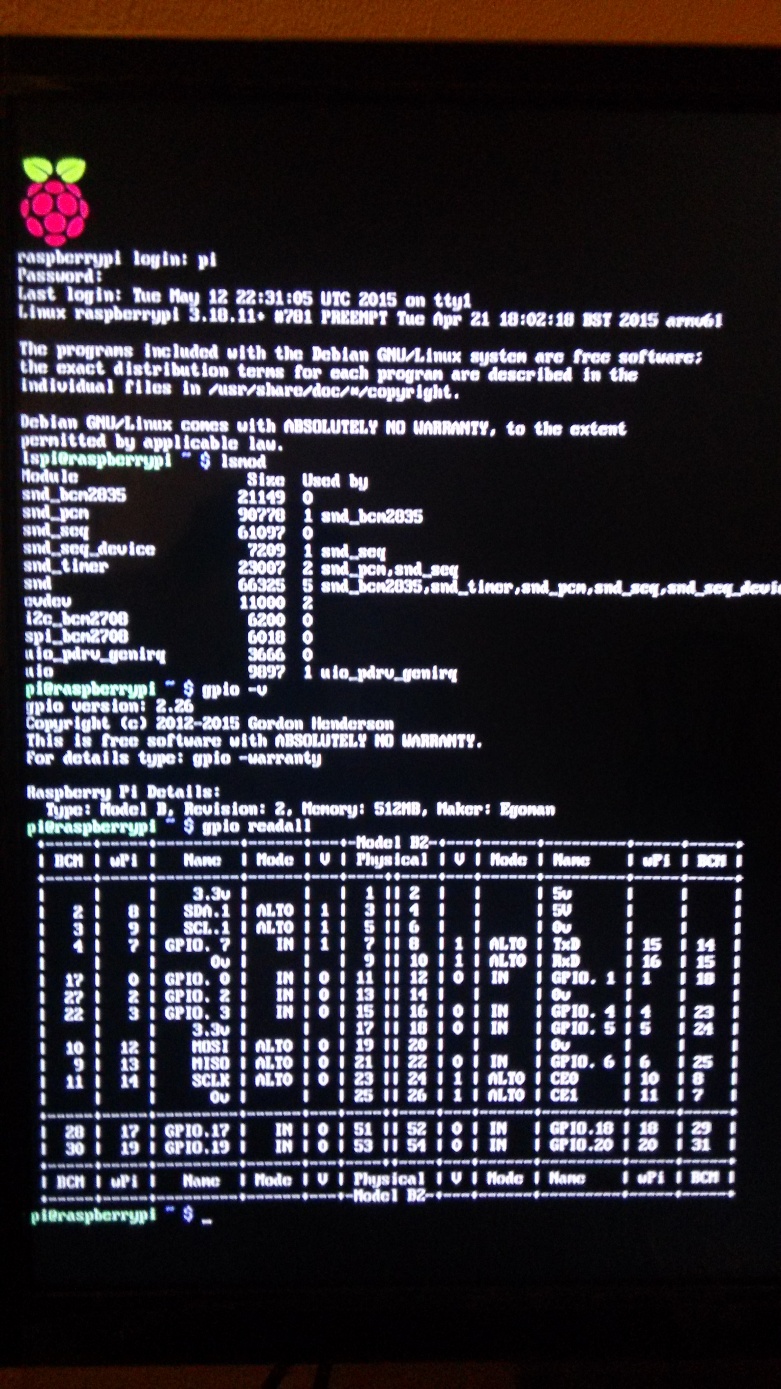
Wire.write(1); // respond with message of 1 byte for synchronization

// as expected by master

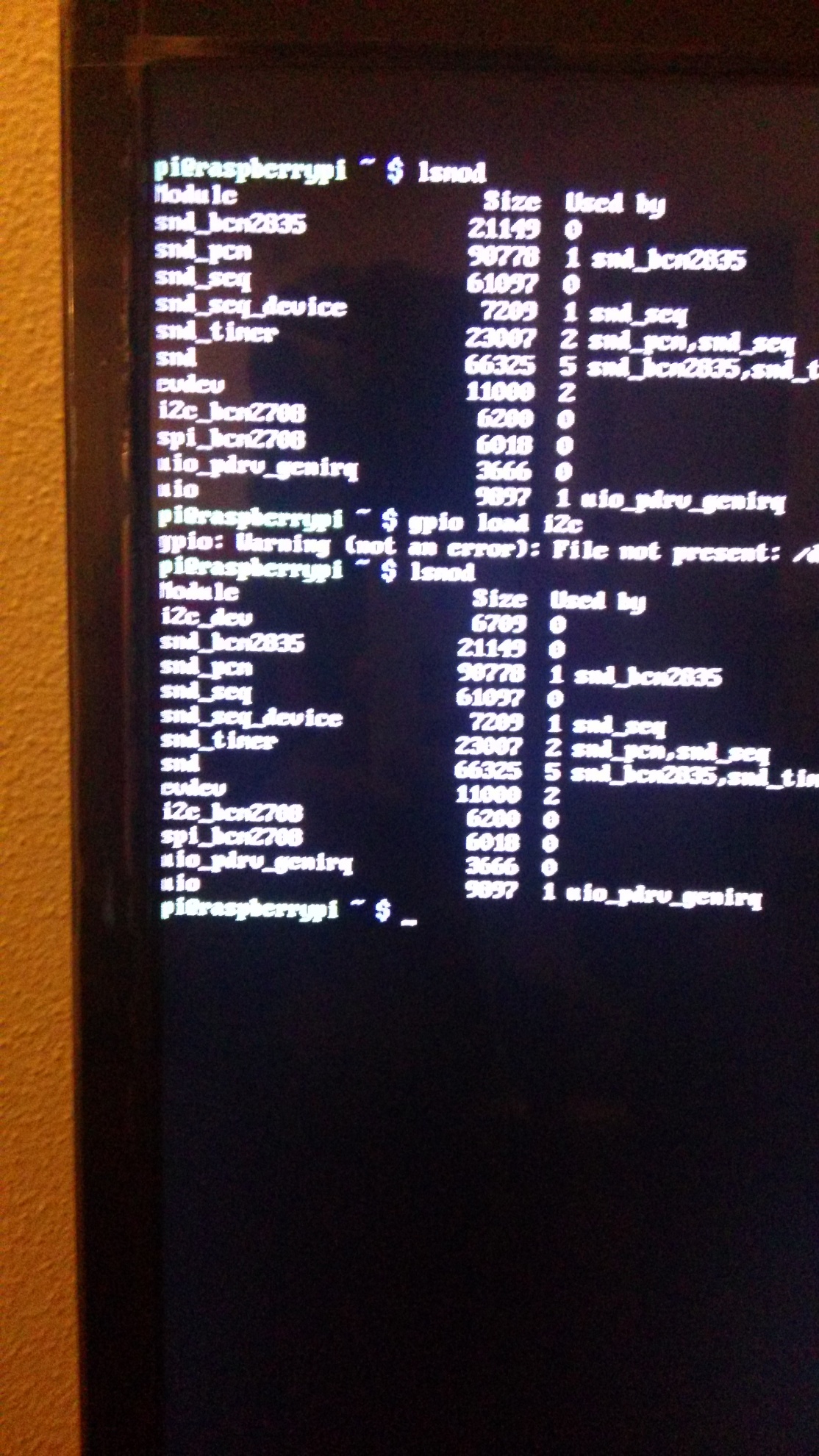
}

The Arduino setup is quite straightforward. Just copy and paste the code and the code will work. However, the Raspberry Pi is really complicated to setup. Here is the step by step:

1. Run sudo apt-get update and finish the update
2. Run sudo apt-get upgrade
3. Reboot the Raspberry Pi with sudo shutdown -r now
4. Check whether the SPI kernel module is installed or not. Run lsmod. The module is installed when spi\_bcm2708 is running.
5. To install wiringPi that is needed for the SPI and I2C communication, we need to install github first. Run sudo apt-get install git-core. Download the wiringPi library using git clone git://git.drogon.net/wiringPi. Then, update it to the latest version by running cd wiringPi then git pull origin. After that, install the wiringPi library by running ./build. Here is the check that all of the modules have been installed.



1. To install the I2C feature, install the helper functions by running sudo apt-get install libi2c-dev.
2. Then, install the kernel module to run I2C by running gpio load i2c.



1. Check the i2c by running i2cdetect –y <number>. <number> is 0 for type A and 1 for type B. in this case, we are running on type B.
2. When compiling the program, don’t forget to add –lwiringPi and –lpthread. The compile command looks like this gcc project.c –lpthread –lwiringPi
3. To run the program, make sure that all of the wiring are correct according to the circuit schematic above. Run the program by calling sudo ./a.out