A black background with a black square

AI-generated content may be incorrect.

Four-Wheel Steering for FSAE Racecar

A report submitted to the University of Arlington’s FSAE team, UTA Racing.

Written By: Osama Almasri, Sarbagnya Bhattaria, Tayden Carter, George Kallam, and Samatar Samatar

May 2nd, 2025

**Abstract**

This report is intended to present the design of Adaptive Steering Solutions approach to electronic four-wheel steering on an FSAE car, specifically F16. The current FSAE rulebook allows 6 degrees of rear wheel steering, and our system is designed to accomplish this using only counter steering. The rear steering geometry and kinematics had to be altered to allow for optimized rear wheel steering given the allowable number of degrees. This required design changes to the cars’ existing components, including new uprights and bell cranks that facilitate the proper connection of components. For system controls, we developed a PD strategy using steering position sensors. This is well suited to adapt to the nonlinear nature of the system with the rear steering varying based on steering wheel angle position.

**Sponsor / Faculty Advisor:**

Dr. Bob Woods - [woods@uta.edu](mailto:woods@uta.edu)

**Team Member Contact Information**

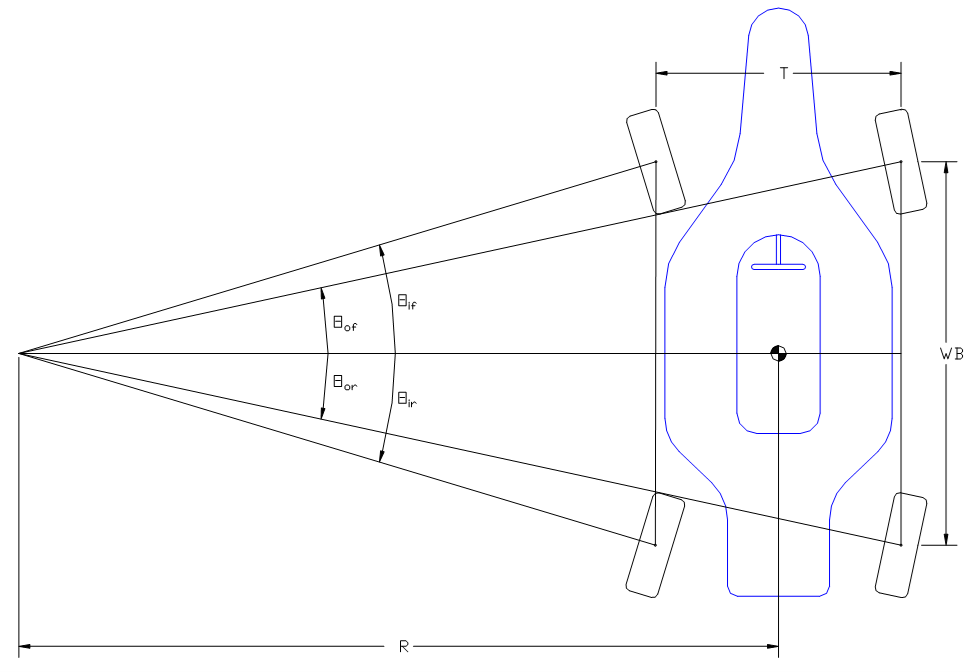
|  |  |  |
| --- | --- | --- |
| **Member** | **Email** | **Phone** |
| Osama Almasri | osama.almasri@mavs.uta.edu | (682) 201 – 6728 |
| Sarbagnya Bhattaria | smb9593@mavs.uta.edu | (817) 542 – 7531 |
| Tayden Carter | [tjc4957@mavs.uta.edu](mailto:tjc4957@mavs.uta.edu) | (817) 889 – 6020 |
| George Kallam | george.kallam@mavs.uta.edu | (817) 229 – 6590 |
| Samatar Samatar | samatar.samatar@mavs.uta.edu | (817) 513 – 1601 |

**Client Information**

|  |  |  |
| --- | --- | --- |
| **Client** | **Email** | **Phone** |
| UTA FSAE | [woods@uta.edu](mailto:woods@uta.edu) | (817) 996 - 0707 |

**Introduction**

Four-wheel steering allows for direct control of all wheels in the vehicle. Most common vehicles today operate in a two-wheel steering system where the front wheels control the direction of the vehicle. The four-wheel steering system can allow for the functionality of two distinct operational modes: parallel steering and counter steering. In parallel steering, all four wheels turn in the same direction which is done at high speeds to leverage the benefits of enhanced vehicle responsiveness and handling. The other mode of four-wheel steering is counter-steer where the rear wheels turn in the opposite direction of the front wheel. This is useful at low speeds. The benefit here lies in greatly reducing the turn radius allowing for tighter turns. This project exclusively focuses on counter-steering; specifically, its functionality in cases of small and large steering inputs.



**Figure 1. Four Wheel Steering Diagram [1]**

The project will see the implementation of the four-wheel steering system onto the UTA F16 FSAE racecar. Although there is no car with an active rear steering system currently, the FSAE department has had a history with four-wheel steering in the past. Particularly in the early 2000’s, attempts were made in this field; one such vehicle exhibited the capability of operating in both parallel and counter-steering modes. However, the transition between the two modes was wildly undesirable leading to a freeze in future projects [2]. In the recent past, the group known as Quad Steering Solutions has designed and developed an electrically driven servo system for the rear . The system that was developed introduced a nonlinear gain and was designed for the purposes of implementation of FSAE racecars.

A close-up of a steering wheel

AI-generated content may be incorrect.

**Figure 2. Test Bed CAD Render**

The focus here at Adaptive Steering Solutions is to implement the previous project's four-wheel steering system onto the F16 race car. The system will be designed in a way such that at small steering inputs the rack will not actuate, but at larger steering inputs the rack will actuate. Additionally, a constraint required for FSAE rules mandates that rear wheels will be limited to only 6 degrees of turning. Before the integration of 4WS can take place, an objective for the project is to develop a functional testbed that will simulate the desired results. The testbed design is seen in Figure 2. In preparation for the system to be implemented on the vehicle, rear suspension components such as bell cranks and uprights must be designed to accommodate the rear steering rack.

**Design**

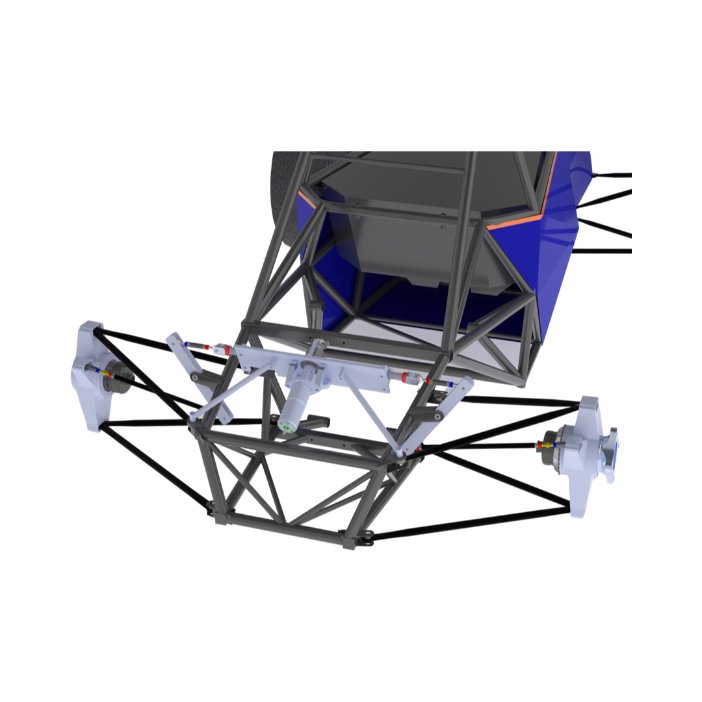
**Design Criteria**

* The Rear Steering system must follow FSAE rules which limit the rear steering angle at the maximum of 6 degrees.
* The system should only actuate at low-speed corners i.e. when the turn radius is small.
* The design needs to be integrated into an already existing F16 race car.

**System Overview**

The cornerstone of the entire system is the electrically driven servo system that actuates the steering rack. For the steering rack to be implemented adjustments to the F16 chassis and suspension components must be made. The complete system is seen in Figure 3 below.

Steering Rack



Upright

Tie rod

Bell Crank

Steering Rack Chassis Mounts

**Figure 3. Rear Steering System Assembly**

The first modification to the system is that the steering rack is placed above chain guards on the drive train. This strategic decision was made to ensure proper packaging and to ensure it works with the location of the tie rod; which was tested for bump steer. (see Appendix C). From here, the steering rack outputs a linear motion which is transferred to the steering rod to the bell crank. The bell crank’s role is to convert the linear motion from the steering rack to the angular motion of the upright. This motion is relayed using the tie rod. The upright’s function is the interface between the wheel hub and suspension links like the steering and control arms. The input of the steering rack will cause the bell crank to pivot which will steer the wheel via the uprights interface. The upright is where the 6-degree constraint is enforced. This system requires kinematic analysis to determine the position of the control arms and tie rod linkages that align with the instant center to achieve minimal bump steer.

**Suspension Modifications**

Bell crank:

Our preliminary design for the bell crank system is shown in Figure 4. A tab is welded onto the chassis where the bell crank is mounted and pivots about. A steering rod linkage connects the steering rack clevis to the bell crank to transfer the motion from the steering rack into the bell crank. Another steering rod linkage, known as the tie rod, attaches the bottom point on the bell crank to the steering arm on the upright. As the steering rack actuates one way or the other, it will push/pull the linkages and turn the upright which will turn the wheels one way or the other.

Steering Rack Clevis

Inner Steering Rod

A close-up of a machine

AI-generated content may be incorrect.

Bell crank

Outer Steering Rod

Upright Steering Arm

Pivot

**Figure 4. Bell Crank System Diagram**

A-Arms and Upright:

The desired kinematics for rear steering was zero caster and king pin inclination, meaning that the upper a-arms and uprights in the rear needed to be redesigned. These values were desired because with the 6-degree steering limit, caster and KPI would have very little affect so having both be zero allows for easier packaging. We fixed all the inboard points and the lower a-arm and only redesigned the upper a-arm moving the ball joint point to meet the desired kinematic values. The lower and upper ball joint points mount to the upright, as well as the tie rod from the bell crank. These three points create the preliminary design for the upright shown in Figure 5.

A close-up of a machine

AI-generated content may be incorrect.

Ball Joint Points

Inboard Points

**Figure 5. A-Arm and Upright Design**

**Electronics/ Controls/ Test Bed**:

With the steering kinematics and design in place the next step was to start designing the testbed. Ultimately the goal for the testbed was to test the electronics and start developing the program that will control the system. We used an ESP32 microcontroller and other electronics shown in Appendix D to control the system along with the servo mounted steering rack. To design the physical test bed, we designed and 3D printed a saddle for the servo to rest in, a 3D printed steering wheel to mount on a potentiometer, and aluminum plate was welded together to mount the potentiometer and steering wheel. The testbed is shown in Fig 5. with all the electrical components as well as servo steering rack.



**Figure 6. Physical Testbed Configuration**

We took all the electronics from Appendix D and mounted them on a protoboard and by doing that had them in a separate enclosure. To run all the wiring throughout the system we decided to use 4-pin Deutsch connectors to have sealed reliable connections through the system. The testbed electronics were designed to be operated on the F16’s battery which is around 13.3V and this was done through a voltage regulator for the Microcontroller and the rest was supplied to the motor driver for the servo.

**Controls:**

The control strategy used in our assembly relies on two potentiometers. The first is mounted to the input shaft of the existing F16 steering rack. This was done using a 3D printed, clamshell type bracket shown in Figure 7.

A diagram of a mechanical device

AI-generated content may be incorrect.

**Figure 7. Front Potentiometer Mount**

This is used as the input to the equations included in Appendix B giving the desired rear steer percentage. This percentage correlates with the 6-degree rear steer maximum constraint.

The actual rear steering position is measured using the second potentiometer that is integrated into our rear steering rack assembly. The controller will determine the error seen between the desired and actual rear steering rack position. Consequently, it will use a PD-type control strategy to drive the servo on the rear rack to appropriately minimize the error. This was tuned on the testbed to minimize the response time, overshoot, and oscillations. Several iterations were tried for PD values and those are included in Appendix B. It is also noted that this will almost certainly need further adjustment when implemented in F16.

**Conclusion**

Adaptive Steering Solutions has designed a system to integrate Quad Steering Solutions electronic servo steering system into UTA Racing’s F16 car. We have reworked the rear steering kinematics and redesigned the rear upper a-arms, bell crank, and upright to meet our desired parameters of zero caster and king pin inclination. Before implementing the electric servo system onto F16, we developed a tabletop testbed to assemble, validate, and tune the system. With this testbed, we were able to get the assembly ready to put onto F16 as it is with all the electronics and controls. Before we finalize all designs and start manufacturing components, analysis of the new bell crank and upright design is still needed.

**Future Work**

With the rear steering kinematics finished, we now have preliminary new designs for the rear uprights and bell cranks. Further analysis and FEA need to be done on these before finalizing designs. The testbed control system has been finished and been prepped for integration into F16 and tune ready. The next semester will see us having the rear uprights and bell crank manufactured and integrated into the car as well as the servo-steering system mounted and tuned to the vehicle. The four-wheel steering system will optimally be tested and calibrated during the summer and be race ready by the end of the semester.

**References**

[1]. Woods, Bob, “4-Wheel Steering Analysis 2022,” February 10, 2022, UTA Internal Documentation, file: “4 Wheel variable steer 2022.docx”

[2]. SAE International, “Formula SAE Rules 2025, Version 1.0,” September 6, 2024, https://www.fsaeonline.com/cdsweb/app/NewsItem.aspx?NewsItemID=379e4a8a-80a2-4a74-87c2-6f2de4212270

[3] Wescott, Tim, “PID without a PhD”, August 14, 2018

**Appendices**

**Appendix A – Turn Radius**

Steering Wheel vs Wheel Angle Test

The circumference of F22s steering wheel was marked every 10-degrees. The 0-reference mark and pointer were aligned at the top of the steering wheel while the wheels were aligned straight ahead. This is shown in Fig. A1. Next, the steering wheel was turned from its centered position at 10-degree intervals until full lock was achieved. Considering this approach the upscale, a downscale measurement was also performed similarly from full lock back to the centered position.

These measurements were taken at each of the front wheels. However, this was performed only in the direction of turning that would cause the measured wheel to be the outside wheel of a turn. This gives us plots of the steering wheel angle vs wheel angle for either front tire, if it's the tire on the outside of the turn.

A close up of a wheel

Description automatically generated

**Figure A1. Steering Wheel with Indicator and Degree Lines**

For the wheel, a large straight metal plate was placed on the side of the left wheel as seen in Figure A2. A string of yarn was placed against the plate and was taped down to the floor. With the car wheels facing forward, the first piece of string represented a 0-degree wheel angle. Next, the driver turns the steering wheel to the left with one tick mark correlating to 10-degree steering angle. The metal plate is then picked up and placed to line up against the now-turned wheel. The string is aligned to the metal plate and tapped down. The process of turning the wheel, placing the metal sheet to serve as a straight edge for the string which will eventually measure the wheel angle. A complete representation of the array of springs is seen in Figure A3. This is repeated until the left wheel is turned fully to the right until it locks.

Once the lock had been reached a high-quality photo was taken to measure the many strings of yarn using the Digital Image Correlation method (DIC); which will be used to measure the wheel angle. After the picture is taken remove the tape and strings to conclude the upscaling measurements. Ultimately this will show the correlation between the steering angle from the steering wheel to the wheel angle from the strings on the ground. Now to begin the downscaling process the wheel is turned back to its initial position, but while placing the metal plate and taping string at each increment.  Once this is complete, take the photo that will be used for DIC for downscaling. Now repeat this same process for the right wheel making a left turn for both upscale and downscale.

A metal square with holes in it

Description automatically generated

**Figure A2. Metal Plate Alignment**

A string of strings on a wooden surface

Description automatically generated

**Figure A3. Wheel Angle String Placement**

The plots displayed below are the data regarding the relationship between steering angle and wheel angle dependent on the turn. Figure A4 depicts a left turn and its corresponding inner and outer wheel measurements, while Figure A5 shows a right turn. It was noted that during the test, the left turn locked at 95 degrees, and during the right turn locked at 105 degrees. The possible reason for this discrepancy is that the steering rack wasn’t centered prior to the experiment. Both Figures A4&A5 represent the averages of upscale and downscale data thus reducing possible errors.

A graph of a car driving

AI-generated content may be incorrect.

**Figure A4. Left turn Inside vs outside Wheel**

A graph of a car driving

AI-generated content may be incorrect.

**Figure A5. Right turn Inside vs outside Wheel**

To calculate the turn radius, we needed a few measurements from the car such as the Wheelbase (WB) and the Front Track Width (TF). The following equation is used to calculate the turn radius of the outer wheel given the wheel angle δ0*𝛿0*. We took measurements of the steering angle and the wheel angles on the way from 0 to 100 degrees and from 100 degrees back down to 0. We did these upscale and downscale measurements on both turning directions to help negate the error. An average was calculated for the wheel angle based on the average of the upscale and downscale measurements of both the left and right wheel.

(1)

In Eq. (1) is the equation we used to calculate the turn radius, and the wheel angle δ0*𝛿0*  is associated with a steering wheel angle input. With the above equation and the wheel angle measurements taken from the car we can calculate the range of the turn radius for both left and right. The Wheelbase for F16 is 66” along with the Front Track Width being 49.5” and Eq. (1) accounts for the values being in inches and converts the output to feet.

A graph of a graph

AI-generated content may be incorrect.

**Figure A6. Turn Radius Going Left**

A graph of a car driving

AI-generated content may be incorrect.

**Figure A7. Turn Radius Going Right**

The plots above show the steering angle and the turn radius at that steering angle, it was calculated using the wheel angle which was measured above. Figure A6 shows the inner and outside wheel steering angle vs turn radius for a left turn. Figure. A7 shows the inner and outsider wheel steering angle vs turn radius for a right turn.

The F22 test was performed because the previous F16 test data had substantial amounts of play in the steering system. It was noted that there was around 5 to 10 degrees of steering wheel rotation. It was observed that the F22 vehicle had less translational and rotational play than its 16 counterpart, but the degree difference for the F22 to reach full lock was approximately 10 degrees. This probably stems from the steering rack not being centered before measurements were taken. Ultimately the turn radius data effectively shows real experimental data of steering wheel angle vs turn radius that is in line with theoretical concepts.

**Appendix B – PID and Controls**

There is an equation that is derived from Dr. Woods data that relates the front turn radius to the rear steering percentage. The Four-Wheel Variable Counter Steering report has all the inputs (Turn Radius) and outputs (Percentage Rear Steer) and the equation that is formed from this table is shown with Y being the percentage rear steer and X being the turning radius.

**Figure B1. Dr. Woods Points for Turn Radius and Rear Steering Percentage**

In the Turn Radius report, we experimentally measured the Steering Wheel Angle vs the Turn Radius for both left and right turns. The turn radius was measured using the outside wheel of each respective turn direction. Also noted is this particular test was performed with the F22 car. The test was initially run with F16, but the data showed some errors. This test will be rerun with F16 after an alignment is performed. Consequently, we have the following equations for the Left Turning Radius and the Right Turning Radius which is shown with X being the turning radius and being the Steering Wheel angle.

Left Turn

Right Turn

In the right turn equation, the degrees are multiplied by -1 in order to make the value positive and easier to read. The centered steering wheel position is considered to be the Top Dead Center (TDC) value. This is the zero reference point from which calculations are performed. The wheel angle is positive CCW from TDC and negative CW from TDC. In the code, this logic is applied, and the percentage rear steer equation is turned into the following. Here, the Y output gives the percentage of total rear steer desired.

Left Turn

Right Turn

Knowing the desired rear steering angle, the rack assembly servo motor is driven to move the steering rack to a desired position. The assembly has a second potentiometer mounted to it. This potentiometer is used to determine the actual rear steering position. The rear potentiometer has 270 degrees of motion making it 135 degrees each way, so the percentage is multiplied by the degrees of the specified turn therefore outputting the desired rear potentiometer angle and controlling the servo-steering rack to make the measured reach the desired. If the percentage of rear steer is 50% then the desired rear potentiometer angle is 67.5 degrees, and the servo will be powered to make the measured potentiometer value reach 67.5 degrees.

A PID control strategy was implemented to appropriately drive the servo to match the desired and actual rear steering position.

PID Tuning

Currently we are only using the Kp and Kd terms of the PID Controller and using the calculated desired rear potentiometer angle compared with the measured actual rear potentiometer angle. Once implemented in F16, the system will likely need to be retuned with the potential for a Ki term being needed there. After testing and iterations using the testbed, the values of Kp = 0.1 and Kd = 0.008 showed the measured rear potentiometer angle reaching steady state earlier than previously tested values.

**Figure B2. System Response Using PD Controller**

Measured Rear Steer Percentage

In the Four-Wheel Counter Steering report from Dr. Woods, there is a table of data that has the Turning Radius and the desired rear steering percentage, and those points were put on an excel plot. The control setup was then formatted to output the Turning Radius and the calculated Percentage rear steer from the above equation and the data was plotted alongside the previous points outputting the following plot.

**Figure B3. Microcontroller Calculated Values Alongside Dr. Woods Data Points**

**Appendix C – Bump Steer Analysis**

When implementing the four-wheel steering system, the suspension systems must be designed in such a way that bump steer is accounted for. The bump steer is when the vehicle wheels move on their own without input from the driver as the suspension compresses and decompresses. To achieve near-zero bump steer, a few criteria must be met. The first is that the location of the tie rod must fall between the upper and lower control arms. Specifically, an imaginary line must run through the endpoint of the upper control arms to the endpoint of the tie rod to the lower control arms. The same logic is applied to the other side of the rods as well. More importantly, the centerline of all three rods must intersect at a location known as the instant center. Both height and length are two predominant factors that dictate how pronounced the bump steer will be.

  The process to achieve an appropriate level of bump steer relied on both SolidWorks and Mitchel kinematics software. SolidWorks contained the master assembly suspension sketch and Mitchel would analyze the sketch and determine the bump steer by examining the toe angle. To obtain relevant results, the SolidWorks coordinate system must be in line with the coordinate system. Next, the suspension system coordinates such as the upper and lower control arms are inserted into Mitchel along with other vehicle information like the wheelbase. Now, on SolidWorks, construction lines must be made connecting the pivot points of all the rods, and centerlines will be constructed so that all three lines intersect at the same point. With proper constraints, a SolidWorks tie-rod configuration can be achieved.

Once the configuration is obtained, the coordinates of the tie rod’s inboard and hub points must be imported into Mitchel’s steering section as seen in Figure C1. After this by selecting compute Mitchel will generate the corresponding bump steer. To compile bump steer data, the roll values must be held to a constant zero while adjusting the ride values (in) from -1 to 1 in increments of .25. It's important to note that achieving near-zero bump steer in the cases for this project is impractical; therefore, a target bump steer value should be 0.05 degrees. The proper location of the tie rod will be identified in an extensive, iterative process where the inboard and outboard points are adjusted until the targeted value is reached.

A screenshot of a computer

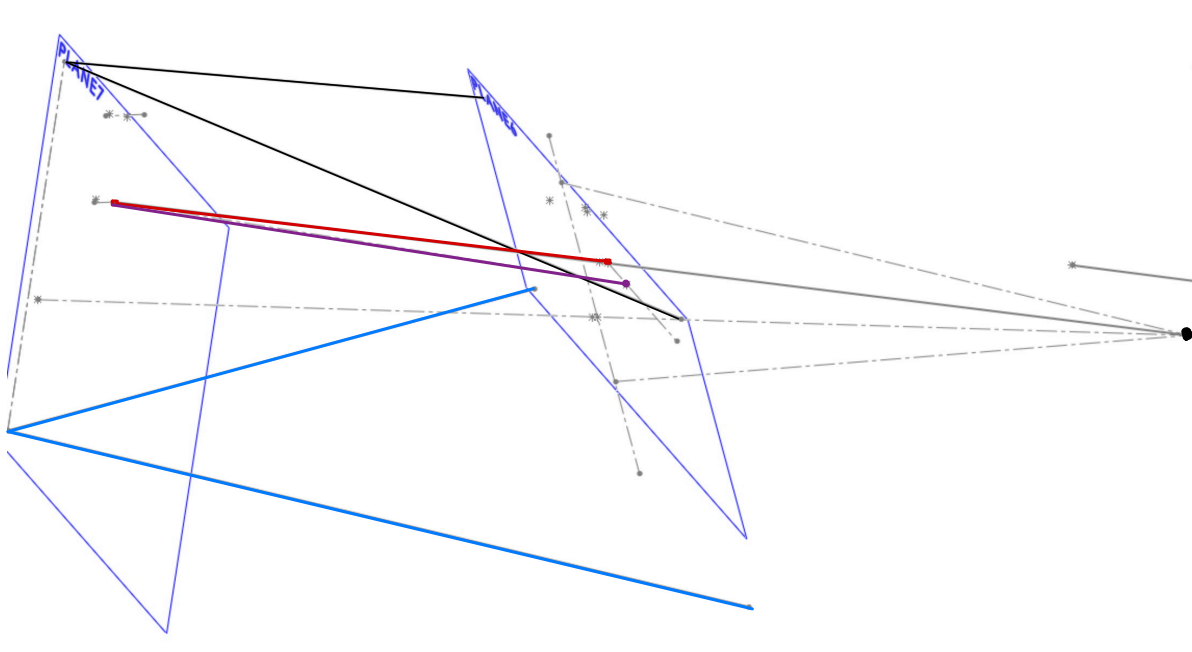
AI-generated content may be incorrect.

**Figure C1. Mitchel Steering Input data**

While finding the point on the tie rod, it was observed that the closer the rod was to the middle the smaller the bump steer values were. However, the first location that was sufficient for Mitchel had complications in SolidWorks from a packaging standpoint as the tie rod was too close to one of the suspension rods. Therefore, it was decided that to fix this issue the location of the inboard point will be moved further outwards in order to prevent packaging issues and maintain an appropriate level of bump steer. The Figure C2 below represents the adjusted configuration showing the purple tie rod inboard point being moved outwards to account for the packaging constraints.

Old Tie Rod

Upper Control Arms



IC Point

New Tie Rod

Lower Control Arms

**Figure C2. SolidWorks Adjusted Configuration Showing New and Old Tie Rod**

In the data that was obtained seen in Figure C3, there is almost a linear relationship between the toe (deg) and ride height (in). On end constraints, point -1 has a toe value of -0.028 degrees and the opposite end has a value of 0.05 degrees. When examining a previous case study on bump steer this relationship shows the tie rod having the correct length but the wrong height. However, as stated previously the goal isn’t to achieve an absolute zero bump steer but to get it as low as possible while considering external factors such as packaging therefore even with this relationship the results are acceptable and will be used in the suspension system.

A graph with blue dots

AI-generated content may be incorrect.

**Figure C3. Bump steer Toe vs Ride Height**

A close-up of several antenna

AI-generated content may be incorrect.**Figure C4. Final CAD Configuration Top and Side View**

**Appendix D – Electronics and Code**

The electronics are designed to be ran off of the FSAE 12V batteries that are used in the car to simplify the system. The Microcontroller that was chosen for this project was the ESP32 and the specs for the controller are in Table D1.

**Table D1. Microcontroller Specs**

|  |  |
| --- | --- |
| ESP32 – DEVKIT – V1 | |
| Operating Voltage: 3.3V | SPIs: 2 |
| Input Voltage: 7-12V | I2Cs: 3 |
| Digital I/O Pins (DIO): 25 | Flash Memory: 4 MB |
| Analog Input Pins (ADC): 6 | SRAM: 520 KB |
| Analog Output Pins (DAC): 2 | Clock Speed: 240 MHz |
| UARTS: 3 | Wi-Fi: IEEE 802.11 b/g/n/e/i |

With the ESP32 Microcontroller we had some previously passed down electronics as this is the continuation of a prior attempt at this project. We were given a Cytron MD13S motor driver that is used to communicate the direction and PWM signals from the microcontroller as well as take in the 13.3V power source from the battery and operate the servo. To have the correct input voltage into the Microcontroller we had to use a L7805 Linear Voltage Regulator and 1µF Capacitors to drop the voltage from 13.3V. For the sensors we also chose to use 10K rotary potentiometers with a linear taper and these were connected to the input shafts of both the front and rear steering racks. We decided to mount all our electronics on a protoboard and do so using a modular design. With all the wires that are connected to our protoboard we are using Deutsch connectors to have a sealed reliable connection and with our potentiometers when it comes to the protoboard we decided to solder a terminal block onto the protoboard for easy troubleshooting. In Fig. D1 we have the protoboard assembly mounted in a printed PLA Enclosure to lower the chance of any short circuiting.

A circuit board with wires and wires

AI-generated content may be incorrect.

Cytron MD13S Motor Driver

L7805 Voltage Regulator

Potentiometer Terminal Block

ESP32 Microcontroller

**Figure D1. Protoboard Assembly in Enclosure**

The electronic components that are used in this project are as follows:

* ESP32 Devkit V1 Microcontroller x1
* Cytron MD13S Motor Driver x1
* L7805 Linear Voltage Regulator x1
  + Mouser #: 511-L7805CV
* 10K Rotary Potentiomer Linear Taper x2
  + Mouser #: 72-BR07230301
* 4 Pin Deutsch Connectors (M-F) x2
* 1 μF Capacitors x2

When this system was assembled, and the electronic configuration was tested and verified to be functional the code layout and structure were designed. The main code aside from the build files was written in C and was structured so that tuning and adjustments for future implementation will be simplified. The code works in the way that it takes in the front steering input from the potentiometer and through the measured equations from the car converts that to a corresponding turn radius. With the turn radius measurement, we can from Dr. Woods equation get the percentage of rear steering and then it calculates the desired rear steering. From the desired rear steering we take the previously mentioned PD strategy and compare the desired rear steering position to the measured rear steering position and increase the duty cycle for the servo to match the desired rear steering position and measured rear steering position. The main.c code is attached below but excludes the header files that contain the cars measured information.

#include <stdio.h>

#include <math.h>     //The C math library that is used for some of the complex math functions needed

#include "driver/gpio.h"    //The header file needed to use GPIO Pins and functions

#include "driver/ledc.h"

#include "freertos/FreeRTOS.h"      //A header needed to use serial communication

#include "freertos/task.h"      //A header needed to use delays

#include "driver/adc.h"     //The header file needed to use ADC pins and functions

#include "sdkconfig.h"     //Will have the pin setups and variable names

#include "TurnRadiusCalc.h"     //The header file for turn radius calc has the coefficient and power values for the function of each wheel

#include "pwm.h"        //The header file where the PWM Values will be stored

#include "RearAngle.h"

double FS\_SteeringAngle; //Input Front Steering Angle, set to -91 for testing the angles, whenever actually implemented this will not have a value

double RS\_SteeringAngle;        //Input Rear Steering Angle

double LW\_Angle;    //Left Wheel Angle

double RW\_Angle;    //Right Wheel Angle

double TR\_Left;     //Turn Radius Left Wheel Right Turn

double TR\_Right;    //Turn Radius Right Wheel Left Turn

double IDEAL\_RS\_ANGLE;

double RS\_TR\_RIGHT;

double RS\_WA\_RIGHT;

double RS\_TR\_LEFT;

double RS\_WA\_LEFT;

double RT\_Percentage;

double LT\_Percentage;

double RS\_Deg;

double deadband = 0;

//double Gain\_Input;    //Not sure if this will be used

//The amount of 'clicks' you want to be able to turn the rear steering adjustment knob

int ADJUSTMENT\_KNOB\_VALUE;      //The value of the adjustment knob that will be used on calculations

#define LED\_PIN 2       //This is for the onboard LED (Status LED)

#define FAN\_PIN 23      //This is for the constant fan to cool the controller

#define FRONT\_STEERING\_POT\_PIN ADC2\_CHANNEL\_7 //This is the front steering pot       ADC1\_CHANNEL\_7 Corresponds to GPIO35 on the pinout diagram

#define REAR\_STEERING\_POT\_PIN ADC2\_CHANNEL\_5 //This is the rear steering pot        ADC1\_CHANNEL\_6 Corresponds to GPIO34 on the pinout diagram

#define ADJUSTMENT\_POT\_PIN ADC1\_CHANNEL\_4 //This is the adjustment pot      ADC1\_CHANNEL\_4 Corresponds to GPIO32 on the pinout diagram

//#define MOTOR\_DRIVER\_1PIN  25      //This is the Motor Driver 1 pin correspodning to GPIO27

//#define MOTOR\_DRIVER\_2PIN 14        //This is the Motor Driver 2 pin corresponding to GPIO14

#define MOTOR\_PWM\_PIN 18        //This is the PWM pin corresponding to GPIO12

#define DIR\_PIN 19

#define ADJUSTMENT\_AMOUNT 7     //The amount of clicks the potentiometer will have, this only needs to be adjusted right here

typedef struct {

    double Kp;

    double Ki;

    double Kd;

    double previous\_error;

    double integral;

} PID;

double Kp = 0.1;        //The Kp value of the PID

double Ki = 0.000;       //The Ki value of the PID

double Kd = 0.008;        //The Kd value of the PID

PID pid;

void setup\_pid(PID \*pid) {

pid->Kp = Kp;

pid->Ki = Ki;

pid->Kd = Kd;

pid->previous\_error = 0;

pid->integral = 0;

}

#define INTEGRAL\_LIMIT 100 // Adjust based on testing

double compute\_pid(PID \*pid, double error)

{

    double P\_out = pid->Kp \* error;

    // \*\*Reset integral when error direction changes\*\*

    if ((pid->previous\_error > 0 && error < 0) || (pid->previous\_error < 0 && error > 0)) {

        pid->integral = 0;

    } else {

        pid->integral += error;

    }

    // \*\*Clamp integral to prevent windup\*\*

    if (pid->integral > INTEGRAL\_LIMIT) pid->integral = INTEGRAL\_LIMIT;

    if (pid->integral < -INTEGRAL\_LIMIT) pid->integral = -INTEGRAL\_LIMIT;

    double I\_out = pid->Ki \* pid->integral;

    double derivative = (error - pid->previous\_error);

    double D\_out = pid->Kd \* derivative;

    pid->previous\_error = error;

    double output = P\_out + I\_out + D\_out;

    // Debug Output

    /\*printf("PID Debug - Error: %lf, P: %lf, I: %lf, D: %lf, PID Output: %lf\n",

           error, P\_out, I\_out, D\_out, output);

    \*/

    return output;

}

int duty\_cycle\_convert(double pid\_output)

{

    int duty = (int)(fabs(pid\_output) \* (PWM\_MAX\_DUTY / 15));

    // Ensure duty cycle is within valid range

    if (duty > PWM\_MAX\_DUTY) duty = PWM\_MAX\_DUTY;

    if (duty > 0 && duty < 300) duty = 400;  // Minimum force to actually move

    return duty;

}

void setup\_pwm()

{

    ledc\_timer\_config\_t timer\_conf = {

        .speed\_mode = LEDC\_HIGH\_SPEED\_MODE,

        .duty\_resolution = PWM\_RESOLUTION,

        .timer\_num = LEDC\_TIMER\_0,

        .freq\_hz = PWM\_FREQ,

        .clk\_cfg = LEDC\_AUTO\_CLK

    };

    ledc\_timer\_config(&timer\_conf);

    ledc\_channel\_config\_t channel\_conf = {

        .gpio\_num = MOTOR\_PWM\_PIN,

        .speed\_mode = LEDC\_HIGH\_SPEED\_MODE,

        .channel = LEDC\_CHANNEL\_0,

        .intr\_type = LEDC\_INTR\_DISABLE,

        .timer\_sel = LEDC\_TIMER\_0,

        .duty = 0,

        .hpoint = 0

    };

    ledc\_channel\_config(&channel\_conf);

}

double READ\_FS\_POT()

{

    int raw\_fs\_val = 0;

    if (adc2\_get\_raw(FRONT\_STEERING\_POT\_PIN, ADC\_WIDTH\_BIT\_12, &raw\_fs\_val) == ESP\_OK) {

        double front\_voltage = (raw\_fs\_val / 4095.0) \* 3.3;

        double fs\_angle = (front\_voltage / 3.3) \* 270.0 - 131.0;

        return fs\_angle;

    } else {

        // Handle error (return 0, NAN, or a special value)

        return 0.0;

    }

}

double READ\_RS\_POT()

{

    int raw\_rs\_val = 0;

    if (adc2\_get\_raw(REAR\_STEERING\_POT\_PIN, ADC\_WIDTH\_BIT\_12, &raw\_rs\_val) == ESP\_OK) {

        double rear\_voltage = (raw\_rs\_val / 4095.0) \* 3.3;

        double rs\_angle = (rear\_voltage / 3.3) \* 270 - 135;

        return rs\_angle;

    } else {

        return 0.0;

    }

}

int POT\_ADJUSTMENT()

{

    int ADJUSTMENT\_RAW\_VAL = adc1\_get\_raw(ADJUSTMENT\_POT\_PIN);  // Get raw ADC value

    int ADJUSTMENT\_NUMBER = (int)((ADJUSTMENT\_RAW\_VAL / 4095.0) \* ADJUSTMENT\_AMOUNT);

    return ADJUSTMENT\_NUMBER;

}

void TurnRadiusLeft(double FS\_SteeringAngle)        //The Equation for the left wheel turn radius

{

    TR\_Left = TR\_LW\_Coefficient\*pow(FS\_SteeringAngle,TR\_LW\_Power);

}

void TurnRadiusRight(double FS\_SteeringAngle)       //The Equation for the right wheel turn radius

{

    TR\_Right = TR\_RW\_Coefficient\*pow((-1\*FS\_SteeringAngle),TR\_RW\_Power);

}

void WheelAngleLeft(double FS\_SteeringAngle)        //The equation for the left wheel angle

{

    LW\_Angle = WA\_LW\_Slope\*(FS\_SteeringAngle)+WA\_LW\_Intercept;

}

void WheelAngleRight(double FS\_SteeringAngle)       //The equation for the right wheel angle

{

    RW\_Angle = WA\_RW\_Slope\*((-1\*FS\_SteeringAngle))+WA\_RW\_Intercept;

}

double IdealRearAngle(double FS\_SteeringAngle)

{

    if (FS\_SteeringAngle > deadband)    //Left Turn Steering Percentage Calc

    {

        LT\_Percentage = -50 \* tanh(0.1 \* (TR\_RW\_Coefficient \* pow(FS\_SteeringAngle, TR\_RW\_Power)) - 4.5) + 50;

        double RearAngle = (LT\_Percentage / 100.0) \* 135.0;

        RS\_Deg = RearAngle;

        return RS\_Deg;

    }

    else if (FS\_SteeringAngle < -deadband)  // Right Turn Steering Percentage Calc

    {

        RT\_Percentage = -50 \* tanh(0.1 \* (TR\_LW\_Coefficient \* pow(-FS\_SteeringAngle, TR\_LW\_Power)) - 4.5) + 50;

        double RearAngle = (RT\_Percentage / 100.0) \* 135.0;

        RS\_Deg = RearAngle \* -1 ;

        return RS\_Deg;

    }

    {

        RS\_Deg = 0.0;

        return RS\_Deg;

    }

}

  void app\_main(void)

{

    // Set up GPIOs

    gpio\_set\_direction(LED\_PIN, GPIO\_MODE\_OUTPUT);

    gpio\_set\_direction(FAN\_PIN, GPIO\_MODE\_OUTPUT);

    gpio\_set\_direction(DIR\_PIN, GPIO\_MODE\_OUTPUT); // Direction control

    setup\_pwm(); // Set up PWM control

    // Set up ADC for potentiometers

    adc1\_config\_width(ADC\_WIDTH\_BIT\_12);

    adc2\_config\_channel\_atten(FRONT\_STEERING\_POT\_PIN, ADC\_ATTEN\_DB\_11);

    adc2\_config\_channel\_atten(REAR\_STEERING\_POT\_PIN, ADC\_ATTEN\_DB\_11);

    adc1\_config\_channel\_atten(ADJUSTMENT\_POT\_PIN, ADC\_ATTEN\_DB\_11);

    // Initialize PID

    setup\_pid(&pid);

    while (1)

    {

        // Read potentiometer values

        FS\_SteeringAngle = READ\_FS\_POT();  // Front steering input

        RS\_SteeringAngle = READ\_RS\_POT();  // Rear steering feedback

        ADJUSTMENT\_KNOB\_VALUE = POT\_ADJUSTMENT(); // Additional tuning input

        // Calculate ideal rear steering angle

        IDEAL\_RS\_ANGLE = IdealRearAngle(FS\_SteeringAngle);

        // Compute PID error (Difference between actual and ideal rear steering)

        double error = RS\_SteeringAngle - IDEAL\_RS\_ANGLE ;

        // Apply deadband

        if (fabs(error) < 1.2) {

        error = 0;

        }

        double pid\_output = compute\_pid(&pid, error);

        // Convert PID output into a PWM duty cycle

        int duty\_cycle = duty\_cycle\_convert(pid\_output);

        // Ensure duty cycle stays within bounds

        if (duty\_cycle > PWM\_MAX\_DUTY) duty\_cycle = PWM\_MAX\_DUTY;

        if (duty\_cycle < 0) duty\_cycle = 0;

        // Determine motor direction based on PID output

        if (pid\_output > 0) {

            gpio\_set\_level(DIR\_PIN, 1); // Move forward

        } else {

            gpio\_set\_level(DIR\_PIN, 0); // Move backward

        }

        // Apply PWM to control motor speed

        ledc\_set\_duty(LEDC\_HIGH\_SPEED\_MODE, LEDC\_CHANNEL\_0, duty\_cycle);

        ledc\_update\_duty(LEDC\_HIGH\_SPEED\_MODE, LEDC\_CHANNEL\_0);

        // Debugging output to verify PID response

        /\*printf("FS: %.2lf, RS: %.2lf, IdealRS: %.2lf, Error: %.2lf, PID: %.2lf\n",

            FS\_SteeringAngle, RS\_SteeringAngle, IDEAL\_RS\_ANGLE, error, pid\_output);

        \*/

        printf("%.2lf\t %.2lf\n", IDEAL\_RS\_ANGLE, RS\_SteeringAngle);

        // LED Blink for Status (indicates loop is running)

        gpio\_set\_level(LED\_PIN, 1);

        vTaskDelay(10 / portTICK\_PERIOD\_MS);

        gpio\_set\_level(LED\_PIN, 0);

        vTaskDelay(10 / portTICK\_PERIOD\_MS);

    }

}