Enhanced Ackermann Steering Platform

Adria Fung
William Parker

April 30, 2015

ENHANCED ACKERMANN STEERING PLATFORM

A Major Qualifying Project Report
submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE, Worcester, MA
in partial fulfillment of the requirements for the
Degree of Bachelor of Science

Submitted to:

Professor Kenneth Stafford (advisor)

Adria Fung

William Parker

April 30, 2015

Abstract

This project examines the capabilities of the Enhanced Ackermann Steering system in comparison to other driveline systems. The Ackermann Steering system, often found in cars, was enhanced, enabling our robot chassis to make zero radius turns. The chassis uses an innovative cam-pulley system that allows for the maneuverability of skid steering without the energy loss due to friction. This driveline system was integrated into a robot chassis that would meet the FIRST Robotics Competition requirements.

Acknowledgments

The Enhanced Ackermann Steering Platform project team would like to thank the following people:

Our advisor, Ken Stafford, for his unwavering guidance and applicable feedback.

Bob Boisse, for his generous contribution of supplies and for letting the team use his shop.

Joe St. Germain, for printing our cam-pulleys and for his supply of VEX parts.

Prof. Michaelson, for letting us make AK212a our second home.

Tracey Coetzee, for helping us acquire materials for the robot.

FIRST Team 190, for letting us borrow their 2011 season robot, 2k11 for driver testing.

Volunteer Robot drivers, for helping us gather test data.

Finally, we would like to thank Worcester Polytechnic Institute (WPI) and the Robotics Engineering Department for giving us an opportunity to apply theory to practice.

Table of Contents

List of Figures

List of Tables

Executive Summary

This Major Qualifying Project examined the capabilities of Ackermann systems in comparison to the skid-steered system. The overall goal of the project was to design an Enhanced Ackermann Steering Platform (EASP) that would be maneuverable at low speeds, stable at high speeds, and could participate in a FIRST Robotics Competition (FRC) game.

A skid-steered driveline is one of the most common drivelines found in robots. It is easy to control, maneuverable at low speeds, and easy to implement. However, one of the biggest issues with a skid-steered robot is that it wastes energy sliding the wheels across the ground. The Ackermann system, often found in cars, allows the wheels to turn about the same turning center. The wheels do not skid laterally during a turn; therefore, no energy is wasted while turning. It is traditionally implemented using linkages, which limit the range of motion.

The EASP used a cam-pulley system to achieve Ackermann steering over 224˚ of motion. The right wheel of the robot was steered linearly from the steering motor. The left wheel steered using the cam-pulleys. These pulleys offset the angle of the left wheel such that the robot maintained the Ackermann condition across its full range of motion. Because of the pulleys, the steered wheels could turn significantly farther than a traditional Ackermann steering system implemented with linkages. This allowed the EASP to turn about the center point between its rear wheels, i.e. a zero radius turn.

The EASP performed a speed test and circle test. It was found that the top speed of EASP is 12 feet per second. The robot also drove within a four-foot wide lane around an eight-foot radius circle with a top “controllable” speed of 7.2 feet per second. The EASP was eventually tested against a skid-steered FRC robot. The obstacle course was designed to test

the low speed maneuverability and high speed stability of both robots. Six robot experienced drivers and seven non-experienced drivers wove the robot around a slalom of cones, around a triangle, and U-turned in a boxed area. It was concluded that of all the drivers, the skid-steered robot was on average, faster than EASP. However, the non-experienced drivers had a faster average time with EASP than with the FRC robot. The robot experienced drivers had a slower average time with EASP than with the FRC robot.

The robot was also tested for energy usage to determine whether EASP was more energy efficient than the skid-steered robot. During the driver trials, the initial and final voltages were taken to determine the difference. However, the team discovered that this method was unreliable as each battery had a different surface charge each time. Therefore, a current test was performed. An ammeter was hooked up to the battery and the current was recorded as each robot turned in place. It was concluded that EASP used 1/10 of the power compared to the FRC robot when robots were turning at 20 revolutions per minute. Therefore, EASP is significantly more energy efficient especially when turning.

The team also drafted several recommendations to improve EASP. The first was to fix the sensitivity issue with the RC controller. It was determined that the RC controller's steering wheel turns about 90 degrees total. However, the steering angle of EASP is 224 degrees. Therefore, it would be beneficial if the angle of the controller's steering wheel matched the steering angle of the robot. Also, one of the robot drivers suggested adding direction indicators on the swerve modules so new drivers would know how far they were turning.

1. Introduction

1.1. Problem Statement

One of the most common types of steering used for robot chassis is skid-steering. In a skid-steering system, the left wheels and right wheels on the robot are driven separately. Figure 1 shows a diagram of a skid-steering system. To drive straight, both sides rotate at the same speed. To turn, one side must drive faster than the other. Skid-steering is often used because it provides high maneuverability at low speeds and is easy to implement. Skid-steered robots are able to turn about the center of their chassis by driving the left and right wheels in opposite directions.

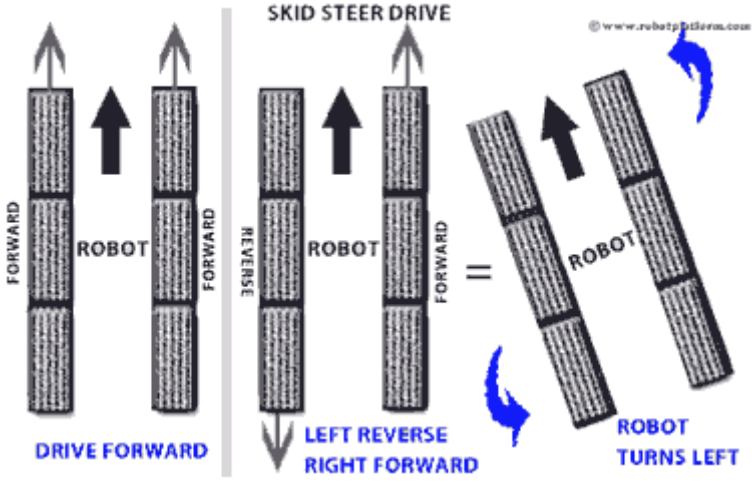


Figure 1: Skid-Steering Diagram [1]

One of the main drawbacks to skid steering is that the wheels must slide along the driving surface during turns. This induces drag, which causes the robot to waste energy. We believe it is possible to design a steering system that has the low speed maneuverability of skid steering and does not cause wheel slipping. To avoid wheel slipping, we look to the principle of Ackermann Steering.

1.2. Ackermann Steering

The Ackermann steering condition occurs when the axes of all the wheels of a vehicle intersect at a single turning point. Figure 2 below shows an example of Ackermann steering compared to parallel linkage steering. In the case of a parallel linkage, the wheels must skid during a turn because the front wheels are turning about different points. By using Ackermann steering, the wheels of a vehicle will never be forced to skid. This increases energy efficiency and reduces tire wear.

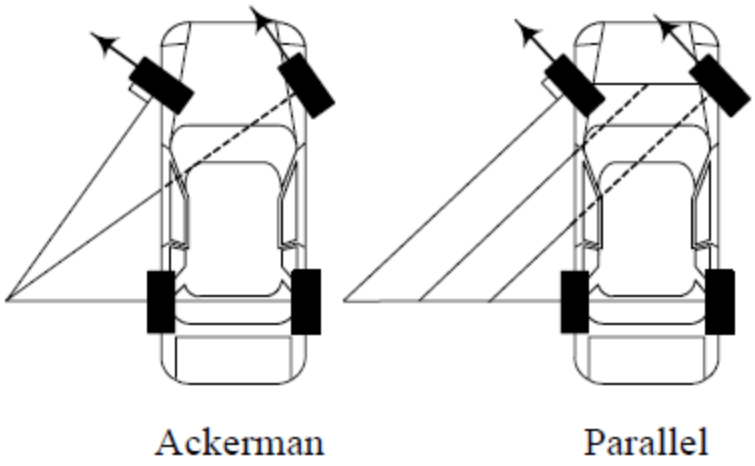


Figure 2: Ackermann Steering vs. Parallel Linkage System [2]

1.3. FIRST Robotics

For Inspiration and Recognition of Science and Technology (FIRST) is a non-profit organization "to inspire young people's interest and participation in science and technology" through robotics involvement. FIRST was founded in 1989 by Dean Kamen and is the umbrella organization for four levels of competition: Junior FIRST Lego League (Jr. FLL) for grades K-3, FIRST Lego League (FLL) for grades 4-8, FIRST Tech Challenge (FTC) for grades 7-12, and the FIRST Robotics Competition (FRC) for grades 9-12.

In the 2015 season, the FIRST Robotics Competition program is projected to have 3000 registered teams with about 75,000 high school students. The FRC Season begins in January. Teams have six and a half weeks to build a robot from a common list of authorized parts provided by FIRST to compete in the current year's game. Because the games are different every year, students, along with their mentors and coaches, are expected to create innovative robot designs to accomplish the given task.

The 2014 game was called Aerial Assist. The objective was to catapult two-foot diameter balls through a goal that was seven-feet above the ground. Along with game rules, the robots' designs were constrained to a set of rules.

To provide reasonable design parameters for this project, the constraints of the 2014 FRC rules were adopted, to include:

- The total length of the frame perimeter sides may not exceed 112in.
- The robot weight may not exceed 120lbs excluding robot battery and bumpers.
- The only motors and actuators permitted on 2014 FRC robots include the following: (part numbers can be found in section 4.7 of the 2014 FRC game manual)

Table 1: Motor selection requirements as stated in the FRC 2014 manual

Motor Name	**Max Quantity Allowed**
CIM	6
BaneBots Motors	5
AndyMark 9015	4
Denso Throttle Control	4
VEX BAG and/or mini-CIM	4
AndyMark PG	3
Window Motors, Door Motors, Windshield Wiper Motors, Seat Motors	2
VEX 2-wire Motor 393	2
Snow Blower Motor	1
Electrical solenoid actuators, no greater than 1 in. stroke and rated electrical input power no greater than 10 watts (W) continuous duty at 12 volts (VDC)	unlimited
Drive motors or fans that are part of a motor controller or COTS computing device	unlimited
Fans included in the 2014 Kickoff Kit, FIRST® Choice, or as a Talon motor controller accessory	unlimited
COTS servos with a maximum power rating of 4W each at 6VDC Per the Servo Industry, Servo Max Power Rating = (Stall Torque) X (No Load Speed)	unlimited

1.4. Project Goals

The overarching goal for this project is to design, build, and test a highly maneuverable robot chassis that utilizes Ackermann steering system enhanced for low speed maneuverability. The chassis must maximize energy efficiency by avoiding wheel slip.

We must also create a driver control system that allows for intuitive control of the chassis. This project is a continuation of a previous project by Michael Cullen et al. [3] titled Optimal Driveline Robot Base (ODRB). The design goals for this project are intended to be similar to the goals of ODRB but with an emphasis on achieving better performance than the ODRB robot. The specific goals are as follows:

- All wheels on the robot are to be driven.
- The robot needs at least four wheels.
- The robot is to be teleoperated.
- The robot must be able to turn about a point that is inside the chassis and along its longitudinal line.
- The error of the steering angles compared to the perfect Ackermann angles cannot exceed five degrees (plus or minus) at all turning angles.
- The robot must be able to go at least ten feet per second.
- While going at ten feet per second, the robot must be able to drive along a ten foot radius circle while staying within a four foot wide lane.
- The robot must complete a course that tests its maneuverability in a time that is on average shorter than the completion time of a 2014 FRC rules-compliant skid- steered robot.
- The robot must also use less energy to complete the course than the skid-steered robot.
- The robot must have 250 square inches of continuous space, from the top to the bottom that is by the steering mechanism. This is to allow room for any mechanisms that could be added should the design be used on another robot.

- The robot will be compliant to the 2014 FRC rules in terms of physical size, weight, and authorized materials used for the driveline.

2. Background

2.1. Optimal Driveline Robot Base Project

The Optimal Driveline Robot Base (ODRB) project started in 2013 with the goal of developing a robotic drive system with the ability to "maintain high speed handling, low speed maneuverability, and maximize energy efficiency by reducing wheel skid [3]." The Ackermann Steering was chosen as the system for the project with modifications to the wheels and tie rod linkage. The extreme proximal pivot point was located at either one of the rear wheels, depending on which direction the robot turned For example, if the robot turned hard right, the turning point would be located at the right rear wheel.

During testing, the robot was capable of zero-radius turning (about either of the back wheels) and was able to maintain the 4-foot lane at full speed when driving around a circle. The ODRB was also compared to a FRC skid-steered robot that complied with FRC rules. The robots were driven along a course and times were recorded. The FRC robot was 1.8% faster than the ODRB [3]. Lastly, it was discovered that the ODRB was more energy efficient than the FRC robot. The average voltage loss for the FRC robot was 1.26 volts; the ODRB's voltage loss was 0.315 volts.

The 2013 project team recommended improvements for the existing ODRB. This project adapted a few of these recommendations. The 2013 project team found that the driver controls were too sensitive and needed to be modified for a robot driver to have better control. The controller used was an RC airplane transmitter. The control system needs to be improved to make driving the robot easier. The turning point of the robot would be more maneuverable if it is between the two rear wheels, rather than on either wheel. The current

ODRB's linkage would "lock up" when turning full left or full right. Therefore, a new system had to be designed to address these issues.

2.2. Steering Mechanisms

2.2.1 Ackermann Linkage

Ackermann Steering is typically achieved using a trapezoidal linkage like the one shown in Figure 3 and Figure 4. Unlike a parallel linkage that would keep both the left and right wheel parallel at all angles, a trapezoidal linkage can approximate the Ackermann condition. However, it is impossible for a trapezoidal linkage to exactly match the Ackermann condition. Also, this type of linkage only approximates the Ackermann condition with proper link lengths.

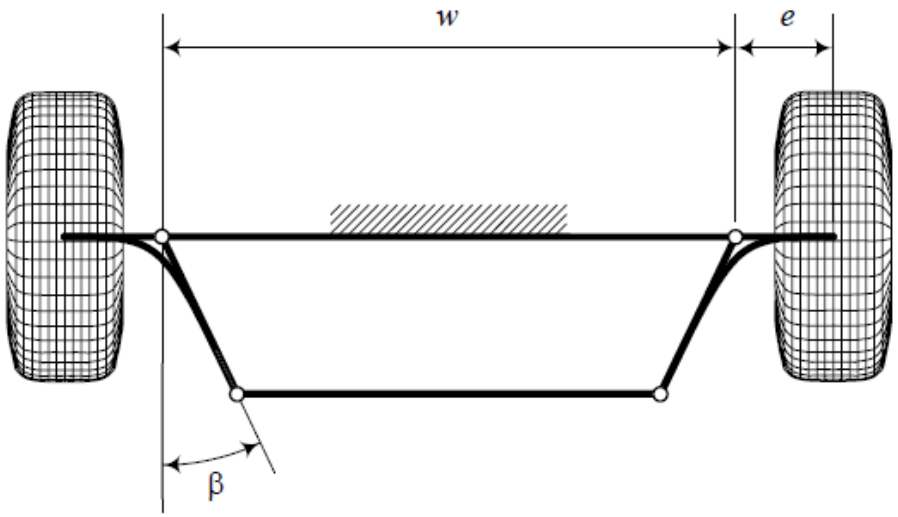


Figure 3: Trapezoidal linkage pointing straight [2]

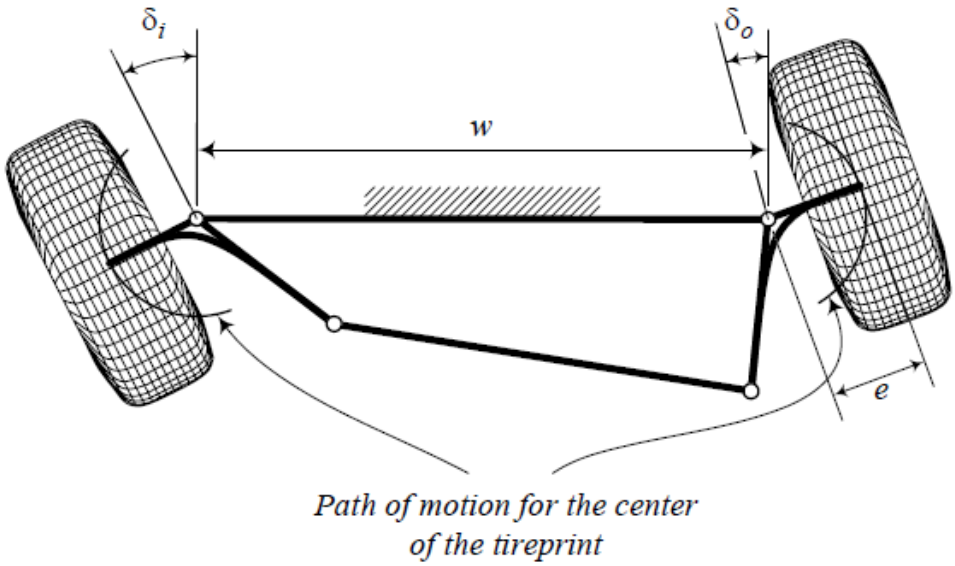


Figure 4: Trapezoidal linkage turning left [2]

Figure 5 shows a graph of the inner wheel angle compared to the outer wheel angle with varying tie rod lengths. By changing the angle β (which changes the tie rod length), the steering mechanism becomes closer or farther from perfect Ackermann. Also, with the best approximation shown in this graph where beta equals 10 degrees, the linkage quickly veers away from the Ackermann condition when the inner wheel reaches 50 degrees. Thus, there is a tradeoff between how closely the linkage follows Ackermann and what range of steering angles can be achieved.

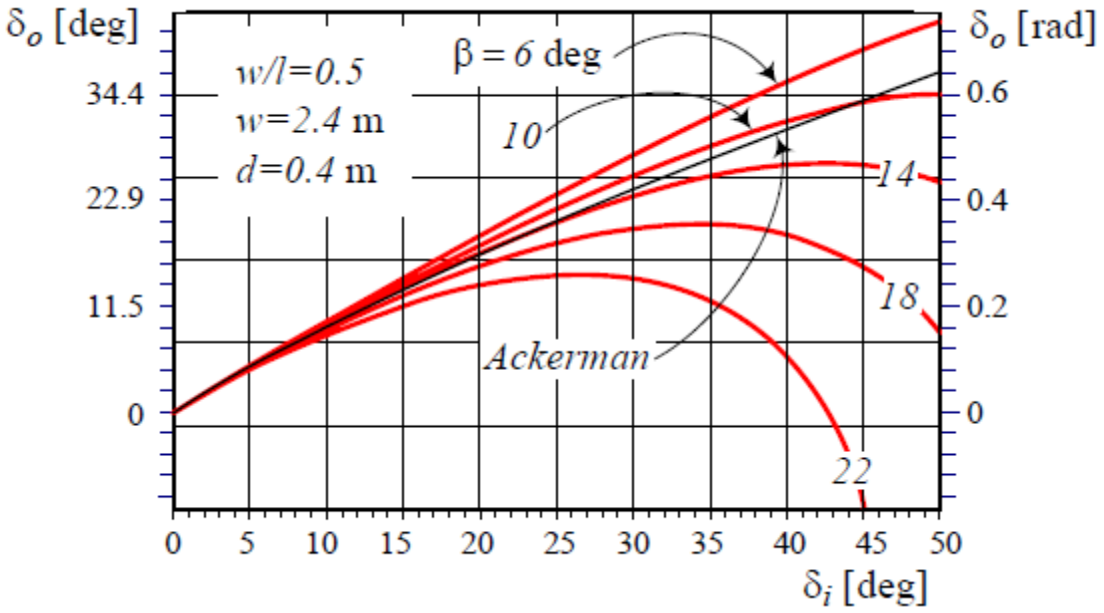


Figure 5: Trapezoidal linkage behaviors compared to perfect Ackermann [2]

2.2.2. Non-circular gears

Non-circular gears are gears that vary in pitch diameter. A diagram of a pair of non-circular gears is below in Figure 6. Unlike typical circular gears, non-circular gears have gear ratios that change as they rotate. This means that the rotational velocity of the driven gear changes as the drive gear rotates a constant velocity. Similarly, the steered wheels in an Ackermann steering system must rotate at continually changing speeds relative to each other in order to maintain the proper angles. Zhao et al. [4] used this principle of non-circular gears to design a pair of "Ackermann gears" that generate the desired Ackermann angles. Though they never manufactured the gears they designed, they did find success testing them in CAD software. Ackermann gears are promising because they can be designed to follow the Ackermann angles perfectly.

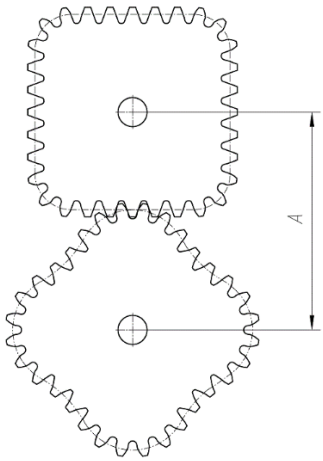


Figure 6: An example of non-circular gears [5]

2.2.3. Swerve Steering

Swerve steering is a steering system where the steered wheels are rotated by motors. In a swerve steering system, each wheel could be individually steered with its own motor, or one motor might steer multiple wheels simultaneously. Ackermann steering could be achieved using swerve steering by having a motor for each steered wheel. Then the robot's microcontroller would have to use feedback from sensors in order to rotate each wheel to the proper Ackermann angle. Like the non-circular gears, this type of swerve steering would theoretically enable the perfect Ackermann condition.

2.2.4. Pros and Cons

Table 2 below outlines the pros and cons of the three proposed solutions. The trapezoidal linkage has too many drawbacks with little benefit, so its consideration as a potential solution did not last long in this project.

Table 2: Pros and cons of three different steering mechanisms

	Pros	Cons
Trapezoidal Linkage	• Simple to manufacture	• Impossible to achieve perfect Ackermann angles at all steering angles • Potential to lock up near toggle position • Input torque required approaches infinity as linkage approaches the toggle position • Takes up a lot of space
Non-circular Gears	• Can achieve perfect Ackermann angles • Compact	• Difficult to design and manufacture*

Swerve Steering	• Can achieve perfect Ackermann angles
	• Potential to be extremely simple to implement (if swerve motors can be direct drive)
	  ○ This would also be compact
		• Requires one swerve motor per steered wheel (at least two would be required given our constraints)
		  ○ each motor would be an additional point of failure
		• Could be difficult to maintain Ackermann angles while switching between two desired steering positions

* To simplify manufacturing, cam-pulleys could be used instead of gears. This would avoid the complexities of designing and cutting teeth for non-circular gears.

2.3. Driver Interfaces

One of the problems from the 2013 ODRB project was that the driver interface was not appropriate for an Ackermann Steering system. The Radio-Controlled (RC) Airplane controller was chosen for inexpensiveness and simplicity. The 2013 project team discovered

that because the driver controls were too sensitive, in a high-paced competition, robot drivers would tend to drive at the extreme range of the turning angle. Therefore, a drive controller that intuitively modeled a car steering and was appropriate for the high-paced competition was most preferred.

2.3.1. RC Airplane Controller

A RC Airplane controller, such as used by the ODRB team (Figure 7) has four channels that control the four functions of an airplane: throttle, ailerons, rudder, and elevator (Figure 8). Although an RC airplane controller emulates the controls of an actual plane, this controller is not preferred for robotics competitions. An RC airplane controller's throttle stick does not have a spring force. Therefore, this allows operators to focus on the 3-axis rotation of an aircraft while maintaining a constant speed in one direction. Because there is no spring force, robot drivers would have to constantly worry about the position of the throttle.

Figure 7: An example of a RC airplane controller

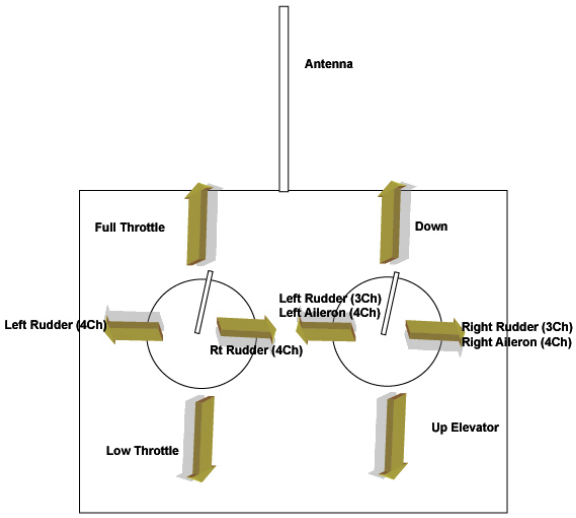


Figure 8: The four channels that control the four functions of the airplane

2.3.2. Steering Wheel Controller

A steering wheel controller (

Figure 9) emulates the steering wheel of an actual car. Steering wheel controllers are ideal for racing games where the operator is in the point-of-view of the driver's seat. This controller is appropriate for Ackermann systems or robots with a car steering functionality. It allows the driver to have better control of the steering capability of the robot. A foot pedal can control the throttle control. However, in FRC competitions, the robot drivers are not in the point-of-view of the driver's seat. Rather, they operate at the ends of the field where they are the 3rd-point-of-view.

Figure 9: A steering wheel controller with foot pedals [6]

For example, in the 2010 FRC soccer game, Breakaway, robots had to maneuver balls into goals located at the same end of the field as the robot drivers. Therefore, if a driver drove the robot straight into the goal, they would direct the controller to go forward, rather than backwards. Figure 10 depicts an illustration of the blue and red teams' scoring robots. The Red Team's robot has to score in the red goal and vice versa for the Blue Team, as indicated by the arrows. The yellow line on each robot indicates the front of the robot. Therefore, if a Blue Team's robot driver were to score in the blue goal, they would need to navigate the robot through a series of reverse controls since the driver's point-of-view is facing the robot head-on. Nonetheless, a steering wheel controller can be appropriate for this competition as long as a team has an experienced driver that is capable of handling the ever-changing orientation of the robot.

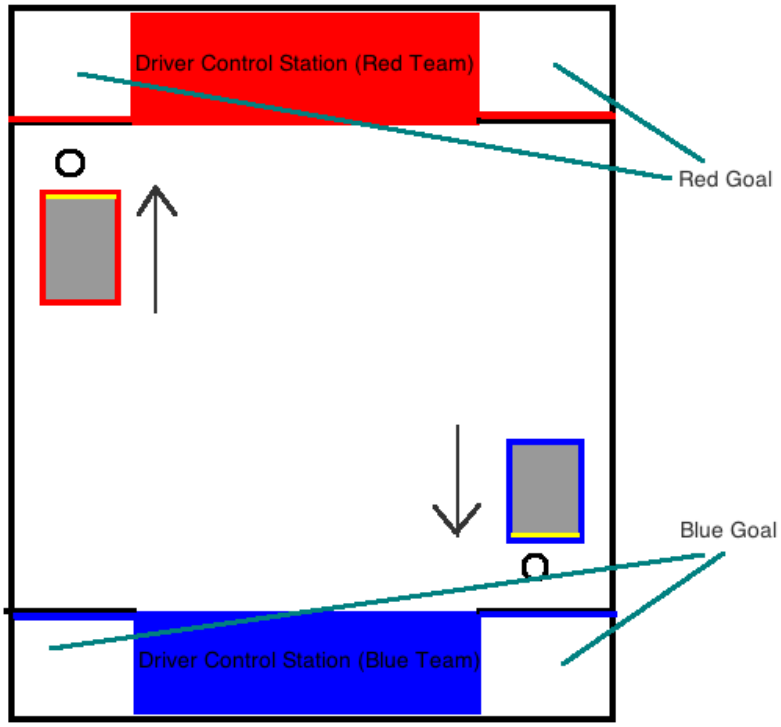


Figure 10: Illustration of scoring robots of 2010 FRC game

2.3.3. RC Handheld Steering Controller

Unlike the steering wheel controller, a RC handheld car controller does not need to be situated on a platform. The controller (Figure 11) is held by the operator with the trigger controlling the throttle and an external miniature steering wheel attachment controlling the steering of the robot. The buttons can be customized to have a two speed transmission. Also, with its small steering wheel, it could difficult to accurately steer the robot. Therefore, the RC handheld steering controller might be difficult to maneuver in tight spaces and around various obstacles in FRC competitions.

Figure 11: A RC handheld car controller [7]

2.3.4. Joysticks

Joysticks (Figure 12) are among the most common robot driving controllers due to their simplicity and functionality. Depending on whether one or two joysticks are used for driving, many different drive systems can be easily driven with joysticks, such as tank drive, arcade drive, and swerve drive. One or wo joysticks can be used to control the Ackermann steering robot, however, it is not intuitive to car steering. Rather than controlling the steering angle using the steering wheel, the steering is controlled by a one-axis linear channel.

Figure 12: Joysticks from the FRC Kit of Parts [8]

3. Methodology

3.1. Independent Swerve Drive Prototype

A swerve drive prototype was made out of VEX parts. This prototype is shown below in Figure 13. The prototype used an Arduino to control it. Only the front wheels were driven since the goal of the prototype was only to see how well the two swerve modules could be controlled to adhere to the Ackermann angles. Each swerve module was controlled using its own PID loop. The Arduino calculated what the outside wheel's angle should be based off of the inside wheel's angle using the Ackermann equation.

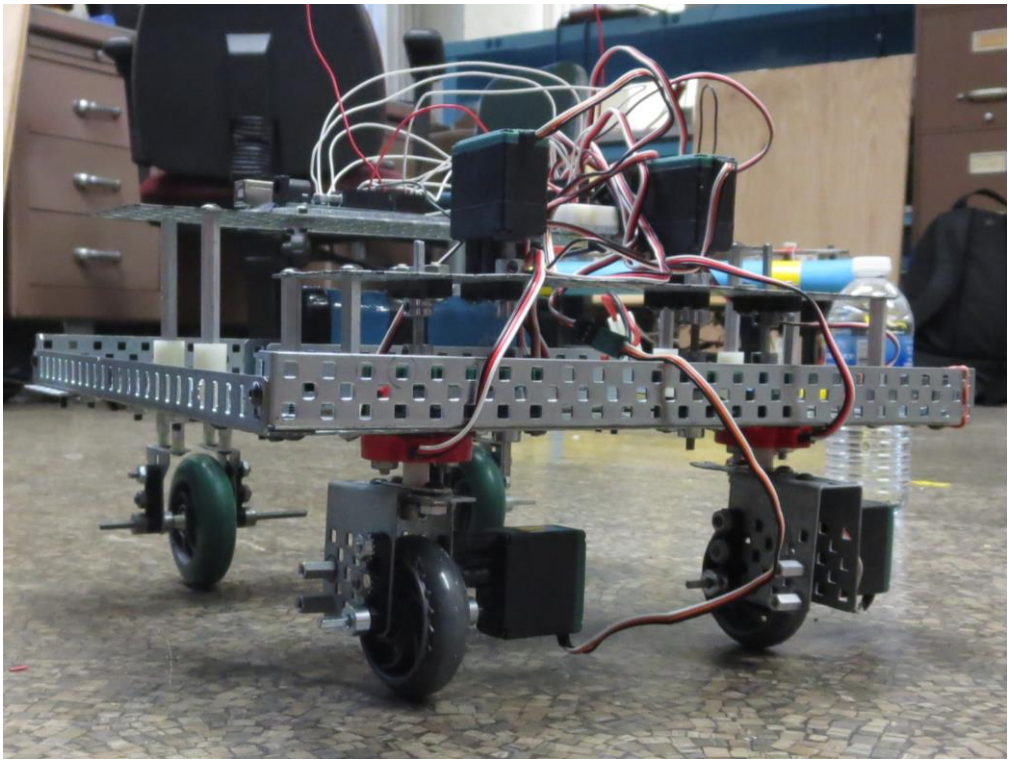


Figure 13: Swerve Steering VEX Prototype

The VEX prototype worked reasonably well. The swerve modules were able to meet the Ackermann condition throughout their full range of motion. The major issue was that

when the wheels were quickly turned, one wheel would be delayed behind the other. This was because the inside wheel got its desired position directly from the remote controller while the outside wheel's position was based off of the current inside wheel's angle. To fix this delay, the outside wheel needed to know where the inside wheel was going, rather than where it currently was. Despite this issue, the prototype showed that this was a viable design for the EASP.

3.2. Cam-Pulley Prototype

Prototype cam-pulleys were made from laser cut acrylic. Multiple layers of the cut acrylic pieces were stacked upon each other to create the pulleys. These can be seen in Figure 14. The design of these pulleys was based on research by Zhao et al. [4] that investigated the possibility of an Ackermann steering system using noncircular gears. Two pairs of cam-pulleys were used so that the steering system would be symmetrical. The contour of the pulleys was not quite correct. The arrow indicators on each pair should have been lined up with one another when oriented straight ahead (as shown in the figure). Also, each pulley was supposed to always be in contact with its mate at all angles.

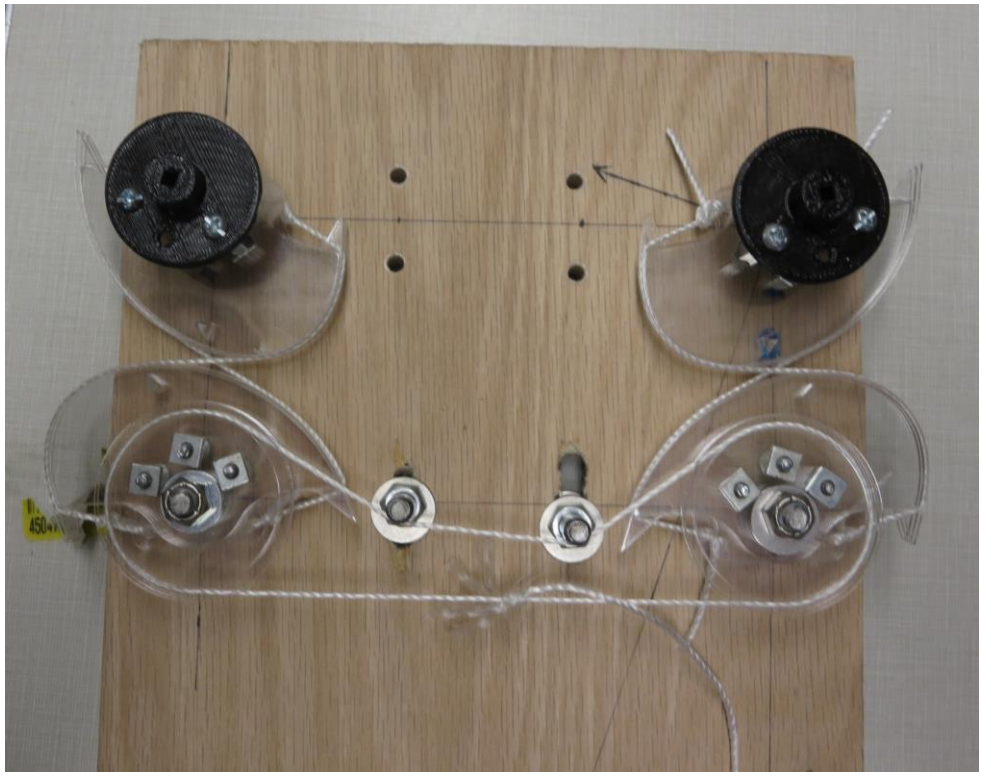


Figure 14: Prototype Cam-Pulleys

The driven pulleys had 3D printed potentiometer mounts attached to them. An Arduino was used to sample the potentiometer readings while the drive pulleys were turned by hand. The data was output over USB to a PC so that it could be imported into Matlab. Figure 15 shows the data collected from the prototype compared to the perfect Ackermann angles. The prototype followed the Ackermann curve reasonably well, but there was noticeable error. There was also some play in the system which caused there to be two slightly different curves depending on which direction the mechanism was being turned. Despite the prototype pulleys not meeting the desired mechanical relationship with each other, they still did a passable job at achieving Ackermann steering across their full range of motion.

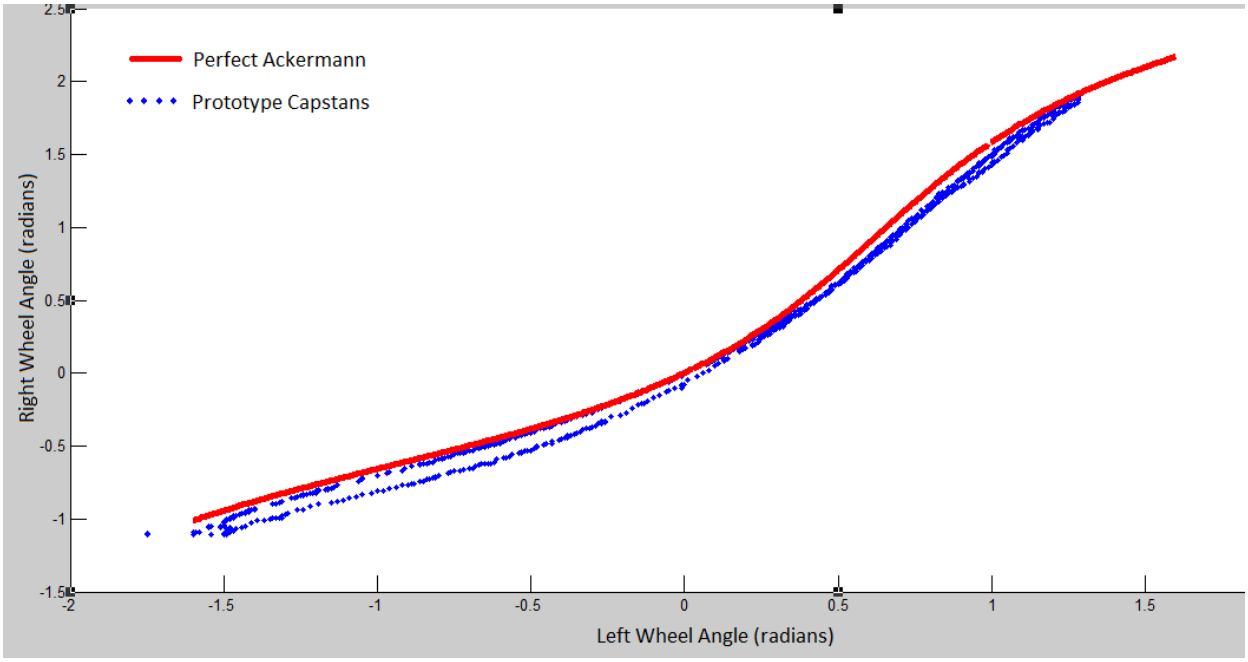


Figure 15: Prototype Cam-Pulley Angles Compared to Perfect Ackermann Angles

3.3. Decision Cost Matrix

Both the cam-pulley system and the independent swerve drive system proved to be viable options based upon the prototypes. To determine which design to utilize, a cost matrix was created that compared different aspects of each system (Table 3). Each metric was rated on a scale of one to five, one being the worst and five being the best.

Table 3: Decision Cost Table

Cost Metric	Steering System	
	Swerve Drive	Cam-Pulleys
Accuracy	4	5
Robustness	3	4
Footprint	5	2
Weight	3	2
Motor Usage	3	5
Center of Gravity	2	3
Software Simplicity	2	4
Manufacturing Complexity	4	1
Innovativeness	1	4
Total Value	27	30

Ultimately, the cam-pulley steering system had a slightly higher score on the cost matrix, so it was the system that was implemented in the final design. The following are the metrics used to compare the two designs and the rationale behind the ratings:

- **Accuracy**: how accurately the steering system could follow the correct Ackermann angles. The swerve drive can closely match the perfect Ackermann angles when stationary, but has difficulty when transitioning from one angle to another. The cam-pulleys – given the correct equation for the contour – would theoretically follow the perfect Ackermann angle exactly.
- **Robustness**: the reliability of the system. The swerve drive has twice as many motors and twice as many feedback loops than the cam-pulley system. Thus it has more points of failure.
- **Footprint**: the amount of space the system would take up on the chassis. The swerve drive would have hardly any footprint if the motors directly drove the swerve

modules. The cam-pulleys cannot directly drive the swerve modules and must be attached to each other via chain or timing belt, increasing footprint.

- **Weight**: the weight of the steering system. The swerve drive would use two motors, but the single motor and cam-pulleys plus their sprockets and chain would probably weigh more.
- **Motor Usage**: how few motors are used. The swerve drive would use two motors while the cam-pulley system would use only one.
- **Center of Gravity**: how low the steering mechanism's center of gravity is on the chassis. The swerve drive, if using direct drive from the motors, would necessitate that the motors be on top of the swerve modules. Since the cam-pulleys are indirectly driving the swerve modules, they could be lower in the chassis.
- **Software Complexity**: the amount of effort that goes into developing the software. The swerve drive system is heavily reliant upon the software feedback loop. The cam-pulley system would also use a feedback loop, but it doesn't need to synchronize the two wheel modules.
- **Manufacturing Complexity**: the amount of effort that goes into creating the mechanism. The swerve drive mechanism would be extremely easy to implement if the motors directly drive the swerve modules. The cam-pulleys would not be easy to manufacture if they are to be machined.
- **Innovativeness**: how innovative and interesting the design is. Swerve drive is a common type of robot steering system. The Ackermann cam-pulley system, as far as we know, has never actually been implemented.

4. Robot Design

The robot chassis developed in this project used the same frame as the previous ODRB project. This includes the swerve modules and drive motors. There were only two significant changes made. The frame was cut to be 3 inches shorter so that the perimeter fit within the 112" limit of the 2014 FRC rules. The resulting wheel track and wheel base were 17" (unchanged) and 20.5" respectively. The other significant change was that the gear ratio for the drive motors was changed from 0.14 to 0.1. This was because the ODRB team incorrectly calculated the gear ratio needed to achieve a top speed of 10 feet per second. An image of the EASP robot is shown below in Figure 16. The rest of Section 4 discusses the design of the Enhanced Ackermann Steering Platform.

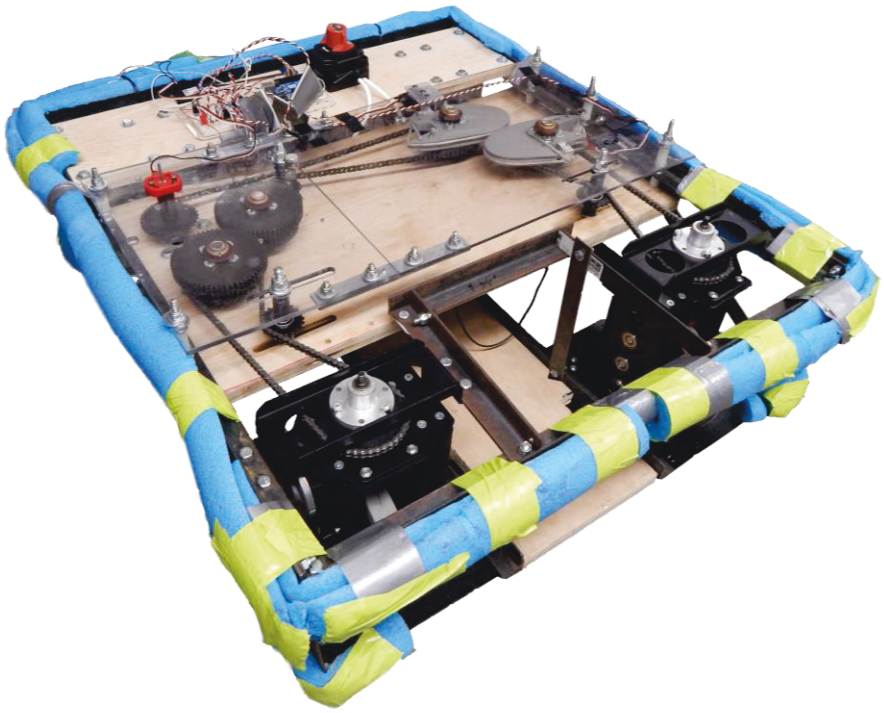


Figure 16: Completed EASP

4.1. Cam Pulleys

The final design for the cam-pulleys used only a single pair rather than one on each side like in the prototype. The downside to this was that the steering system was no longer symmetrical. However, it was mathematically simpler to derive the equations for the cam contour. The equation that defines the contour is

$$r = \frac{d}{\left(\frac{t}{b}\right)^2 * \sin^2(\theta) - \frac{t}{b} * \sin(2\theta) + 2}, \quad (1)$$

where r is the radius of the cam, d is the distance between the centers' of rotation of the two cams, t is the wheel track, b is the wheel base, and θ is the independent variable. This equation was derived by Zhao et al. [4]. In the case of the EASP, the wheels must rotate such that they can turn about the center of the rear wheels. Thus, they must turn 90° plus or minus some offset φ depending on which direction the wheel is turning (Figure 17).

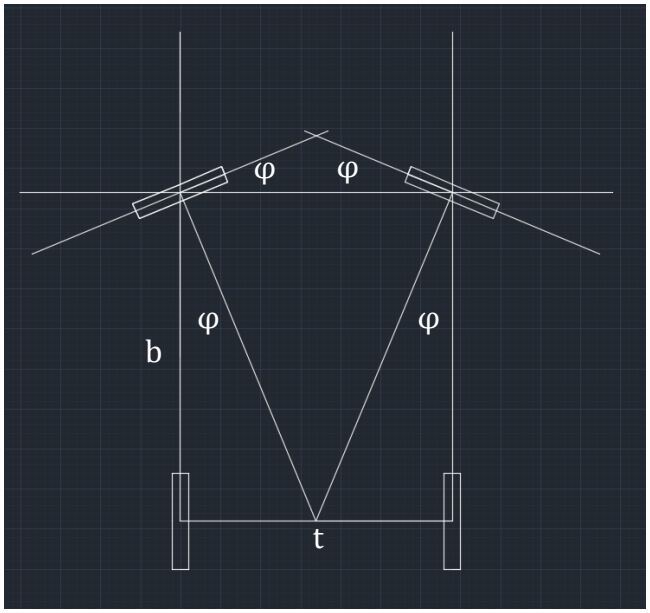


Figure 17: Angle Offset at Turning Extremes

From the diagram above, angle offset φ is

$$\varphi = \arctan\left(\frac{t}{2b}\right). \tag{2}$$

When evaluated by substituting 17 inches as the wheel track and 20.5 inches as the wheel base, φ equals 22.5˚. This means that the contour of the cam is polar equation (1) when θ is swept from -112.5˚ (-90˚ - 22.5˚) to 67.5˚ (90˚ - 22.5˚). A plot of the cam contour was generated in Maple and is shown below in Figure 18. The point at which the two cam-pulleys would meet when the robot is pointing straight is at θ = 0˚, where the radii of the two cams are equal (2.5 inches).

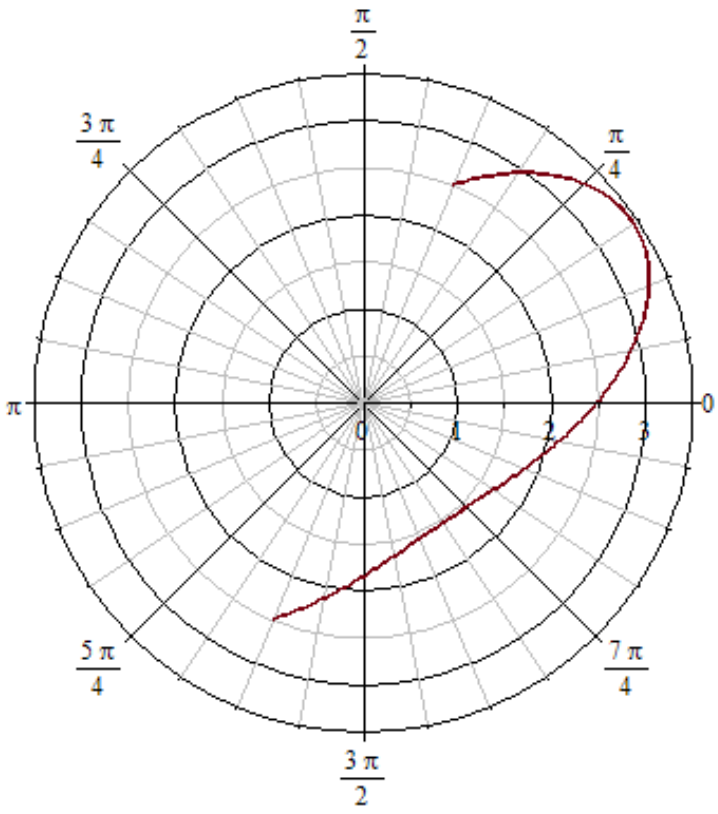


Figure 18: Cam Contour Polar Plot

A final prototype was made with this contour, once again using laser cut acrylic. After verifying that the cam-pulleys mated properly, work began designing the actual pulleys in Solidworks. One parameter that had to be determined was the distance d between the mating cams. The smaller d was, the smaller the pulleys would be, which would result in a smaller footprint. However the pulleys had to be large enough that some sort of tensioner device for the pulley cords could fit on them. The acrylic prototypes used d = 3 inches, which was too small, so the first CAD design used d = 4 inches. When trying to fit a tensioner onto the pulley, it was determined that 4 inches was still too small, so d was increased to 5 inches.

Figure 19 and Figure 20 show the top view and an isometric view of the final CAD model for the cam-pulley. The pulley is bolted to a sprocket that is connected either to the drive motor or a swerve module. The pulley has slotted holes for the sprocket bolts. This allows each pulley 20˚ of rotation for alignment purposes. The cables are tensioned using a tensioner block. One end of the cable is tied to one pulley's tensioner block, while the other end is tied to one of the mating pulley's tie-down holes. To adjust the tension, the block is moved up or down the ¼"-20 bolt that it rides on. The block has about 1.25" of motion. The two small holes at the bottom (Figure 19) are for 8-32 screws. These screws are used as bumpers to press limit switches.

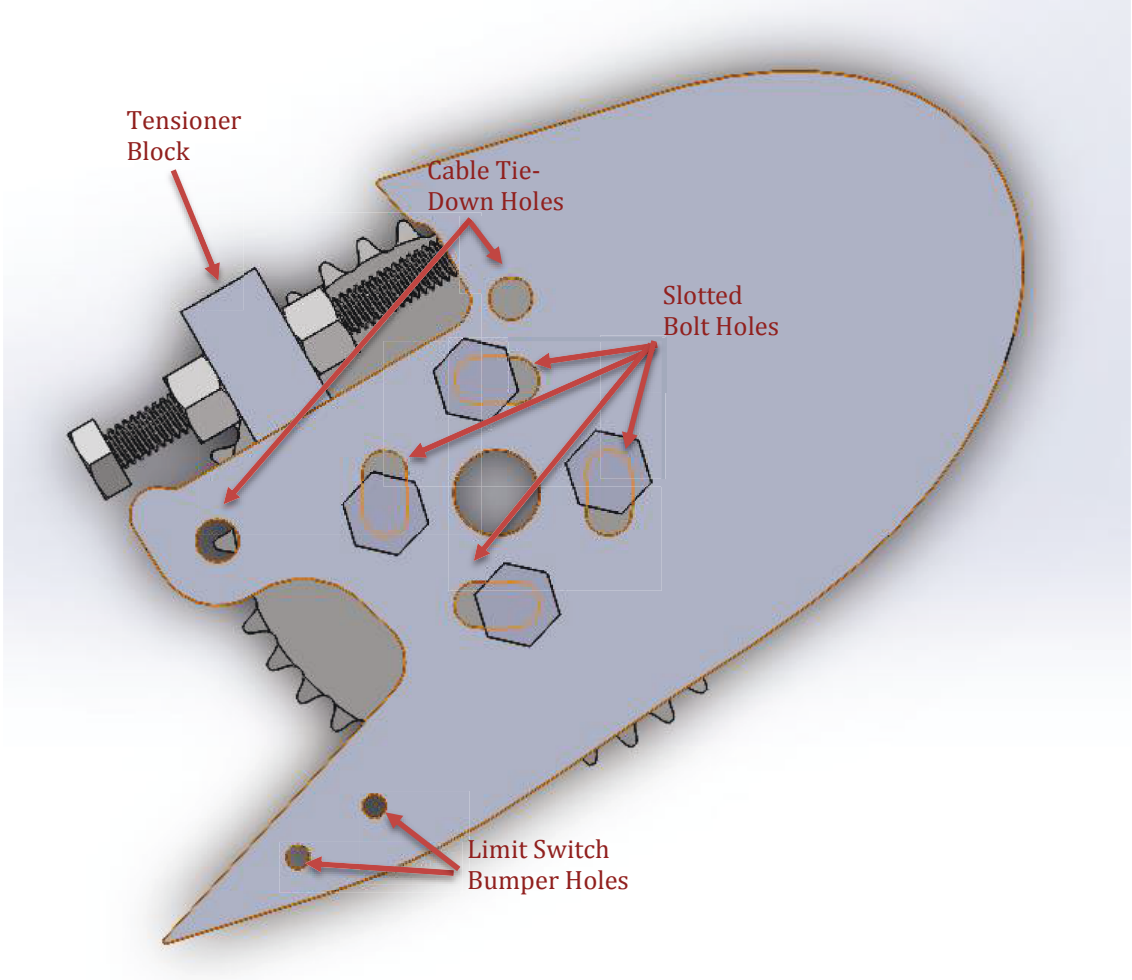


Figure 19: Top View of Cam-Pulley

Two 1/8" wide, 1/16" deep slots are cut into along the cam-pulley's contour. These are for the two pulley cables to fit into. The maximum cable diameter is 1/8". The pulley rides on a ½" axle. The pulley is fixed to the axle via the sprocket, which has a set screw to grab the axle.

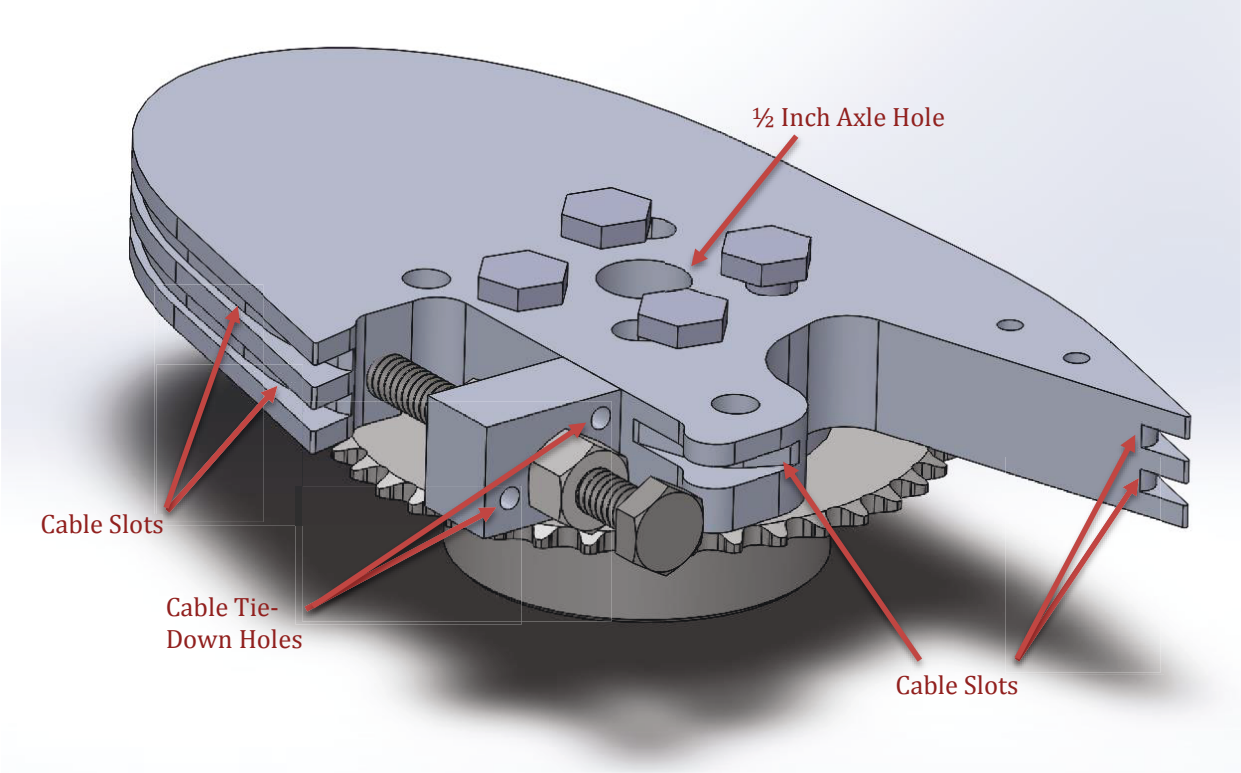


Figure 20: Isometric View of Cam-Pulley

The cam-pulleys were originally intended to be machined from aluminum. However, due to lack of the proper tools, there was no way to cut the slots or to drill the tap hole for the tensioner bolt. Instead, the parts were 3D printed out of ABS plastic. Since the pulleys had to withstand high torques, the printed parts were filled completely, rather than using an internal lattice structure that is typical of 3D printed parts. Even with the solid fill, it was not anticipated that the printed pulleys would be strong enough to be a permanent solution. However, after many hours of use, the cam-pulleys have had no issues and show no signs of structural failure.

4.2. Overall Steering Mechanism

The cam-pulleys, while an important part of the steering mechanism, were only a part of the system as a whole. Figure 21 shows a CAD model of the whole steering mechanism. A Bosch van door motor powers the mechanism. It turns a 30 tooth sprocket (#25 chain) that powers the drive gear and drive cam-pulley (both with 40 tooth sprockets). The gear ratio between the two large gears is 1:1. They are used only to reverse the direction of the right swerve module so that it turns the same direction as the left. The cams offset the angle of the left swerve module so that the Ackermann condition is met.

There are two types of feedback to control the mechanism. A VEX potentiometer attached to the van door motor is used to set the desired angle of the wheels. An adapter was 3D printed to go from the 7/16" diameter van door motor shaft to the 1/8" square VEX axle. The other feedback comes from two VEX limits switches. The cams use 8-32 screws as bumpers to press the switches. These switches stop the mechanism from turning too much, which could cause damage the cam-pulleys.

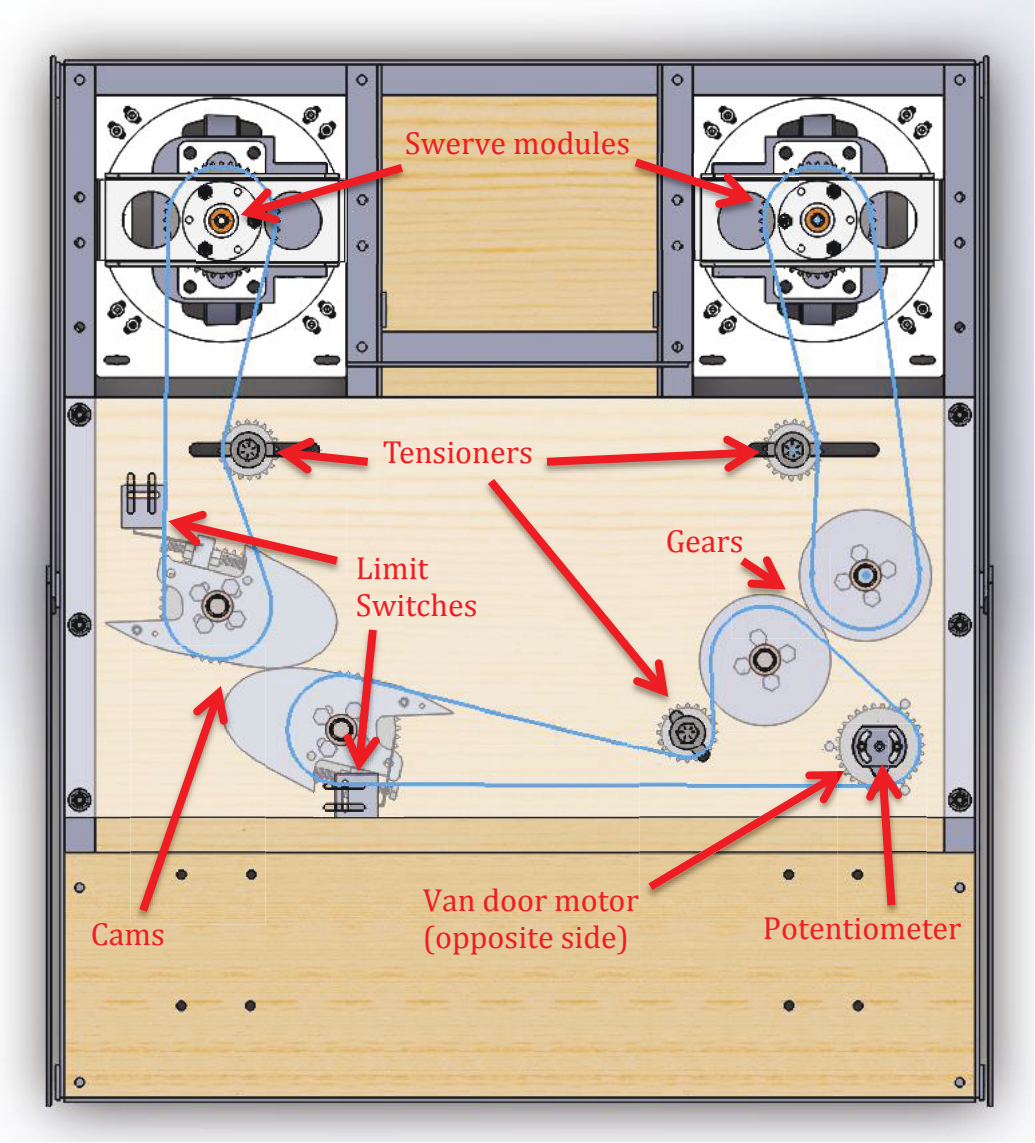


Figure 21: Top View of Entire Steering Mechanism

The van door motor was chosen because it enabled the robot to achieve the goal of a one second turn from one extreme to the other (180˚ for a single swerve module). This is ½ a revolution per second, or 30 RPM. The torque required to turn a swerve module on carpet was 2.22Nm. This value was measured by attaching a lever to a swerve module and pulling perpendicular to the lever with a force gauge. The force needed to turn the module was 7N and the lever was 0.3175m. Because there are two swerve modules the total torque

necessary to turn them both is 4.45Nm. Assuming an efficiency of 80% due to the chain, the required input torque becomes 5.56Nm. The van door motor is capable of about 42 RPM at this torque [9], which meets the 30 RPM goal. However, to reduce the strain on the motor we use a sprocket gear ratio of 3:4. This means that the motor only needs to output 4.17Nm (5.56Nm * 0.75). At this torque, the motor spins at roughly 43 RPM, resulting in an output speed of 31.4 RPM.

4.3. Circuitry

The circuit diagram for the robot is shown below in Figure 22. The robot is powered by a 12V lead-acid battery. Five Victor 884 motor controllers are used to control the four CIM motors and the Van Door motor. A 5V voltage regulator supplies power to the Arduino. The Arduino is connected to two kill switches: one is connected to the 5V peripheral power line and the other to the peripheral ground line. This is to stop the robot if any of the feedback peripherals like the potentiometer or limit switches somehow lose power.

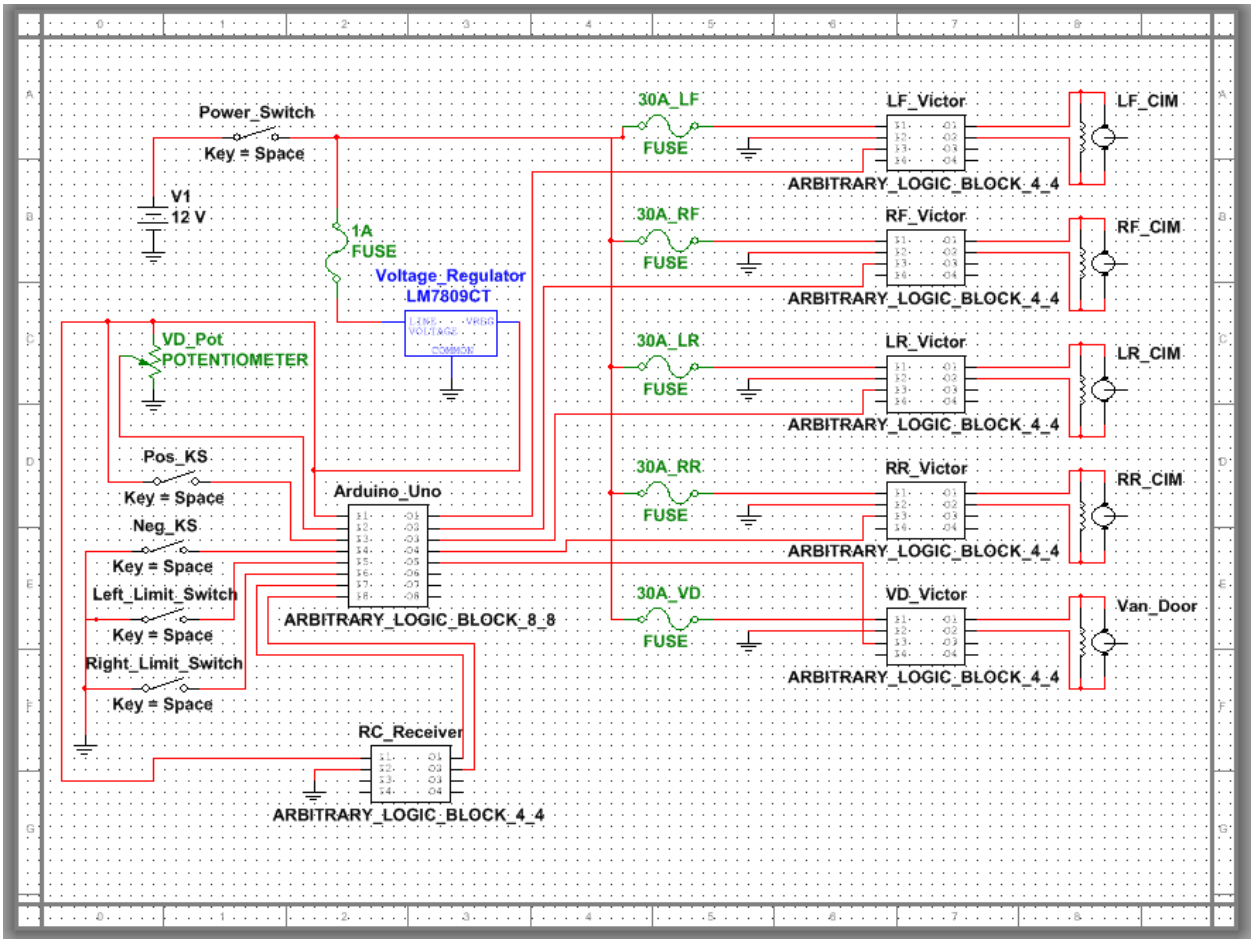


Figure 22: Circuit Diagram

4.4. Software

The top level program flowchart is below in Figure 23. The code gets the controller inputs and updates the steering angle and wheel velocities periodically. If the RC receiver is getting atypical signals, it is a sign that reception was lost. This causes the Arduino to perpetually reset itself until reception is regained. This prevents the robot from going out of control when reception is lost. The Arduino will also stay in a reset loop if any of the kill switches are triggered.

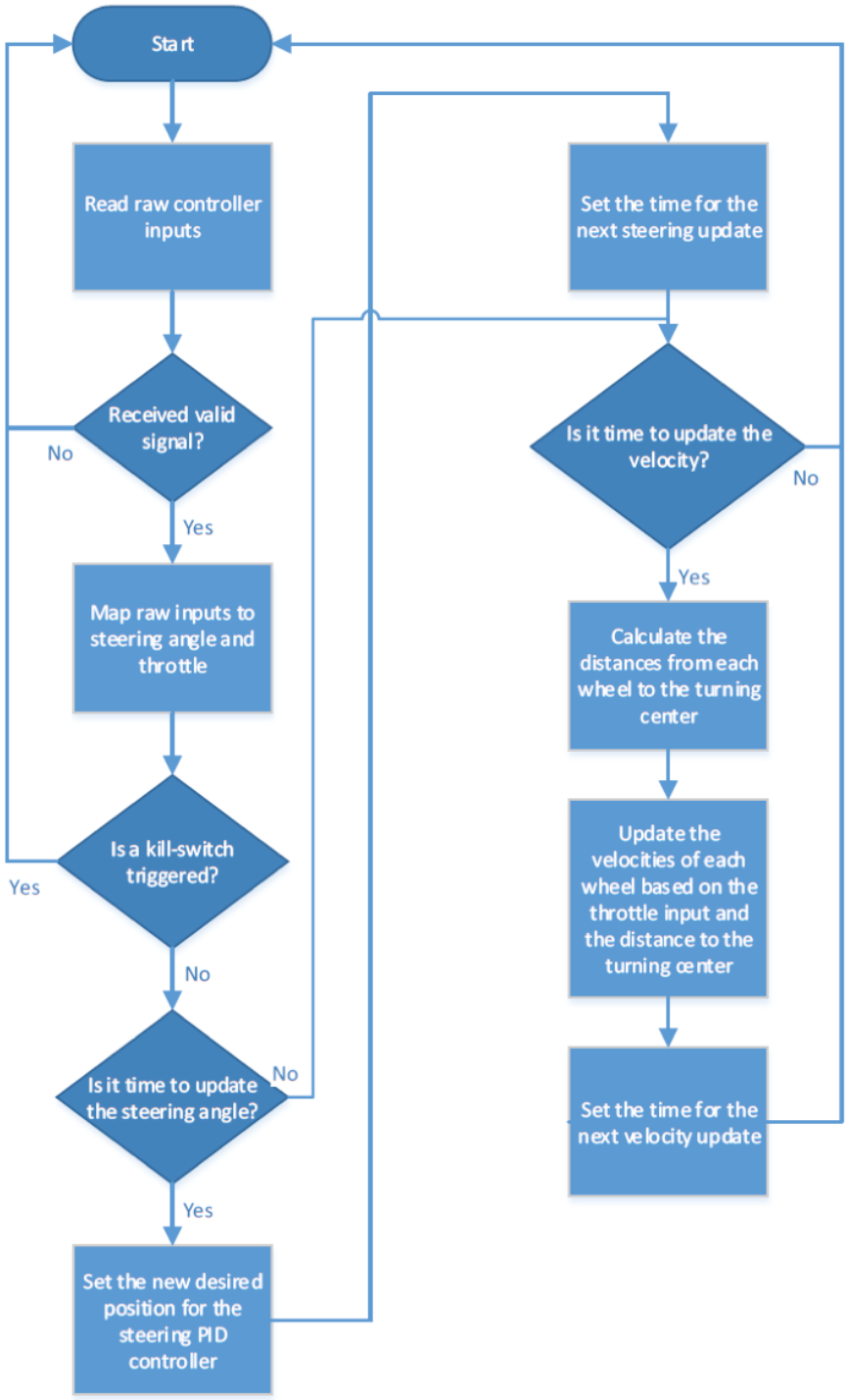


Figure 23: Top Level Program Flowchart

The steering system uses PID to set its position. The desired position is set periodically using a one millisecond resolution clock. The desired steering angle is mapped

from the raw controller input to a value between -112˚ to 112˚, the min and max angles of the inner wheels (the wheel that is on the inside of a turn) of the steering mechanism. The desired potentiometer value is calculated from this angle. This calculation is different depending on if the robot is turning left or right due to the asymmetry of the steering mechanism. When turning right, the inner wheel angle is -¾ times the van door motor angle due to the 3:4 ratio of the van door sprocket to the gear sprocket and the direction reversal from the gears. When turning left, the cam-pulleys are controlling the inner wheel angle, so the Ackermann equation must be used to calculate the necessary potentiometer value for a given angle.

The motor velocities are not set using PID. Instead, the PWM signal to the motor controllers is set directly. The velocity of each wheel must be set individually since the wheel speed is dependent upon the wheel's distance to the turning center. In a turn, the wheel farthest from the turning center has its velocity set to the mapped throttle value. All the other wheel velocities are some fraction of that value depending on how much shorter their turning radii are. In the case of the two rear wheels, one of their turning radii can actually become negative when the robot is turning about a point within the robot. This causes the wheel to change its direction.

4.5. Drive Controller

The device used to control the robot is the RC car remote shown in Figure 24. It is a two channel transmitter, one channel for throttle and the other for steering. This type of controller was chosen because it is used by RC car hobbyists, and due to the Ackermann steering system EASP uses, it drives similarly to a car.

Figure 24: RC Transmitter

5. Test Plan

5.1. Speed Test

One of the project goals stated, “the robot must be able to go at least ten feet per second.” In order to determine the speed of the robot, markers were taped on ground at one-foot increments, as shown in Figure 25. The team videotaped the robot driving at full speed along the markers and looked back on the footage to count how many markers the robot passed at one second.

Figure 25: Taped Markers at one-foot increments

5.2. Circle Test

Another project goal stated, “while going at ten feet per second, the robot must be able to drive along a ten-foot radius circle while staying within a four-foot wide lane. Due to the size constraints of the carpet and area on the testing floor, an eight-foot radius circle was marked with a four-foot wide lane, shown in Figure 26. The team drove the robot along the circle at close to full speed.

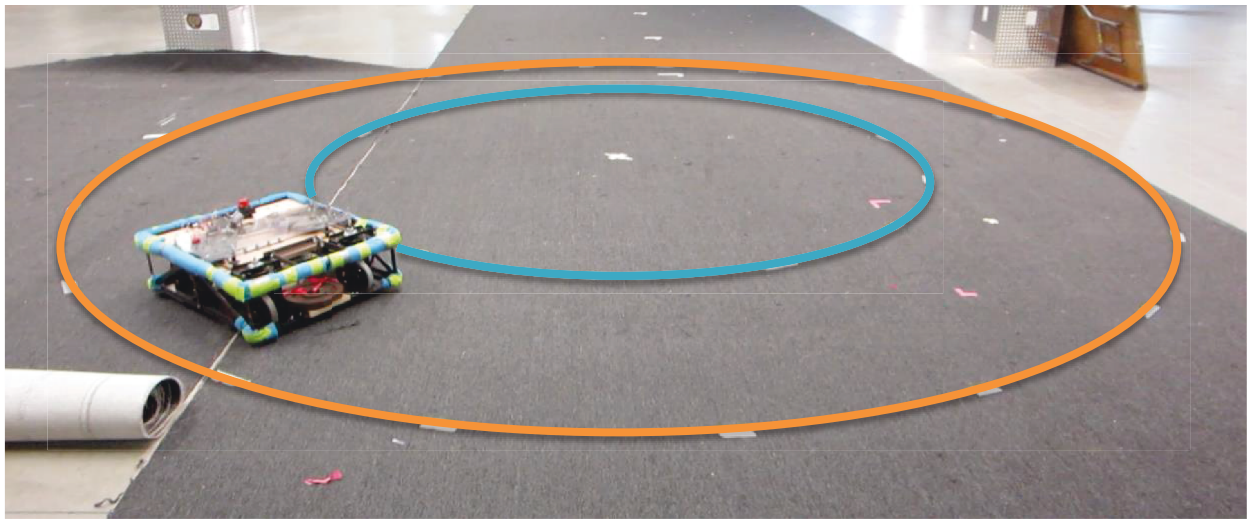


Figure 26: Eight-foot radius circle with four-foot wide lane

5.3. Skid-Steered Comparison

Two of the most important project goals stated that, "the robot must complete a course that tests its maneuverability in a time that is on average shorter than the completion time of a 2014 FRC rules-compliant skid-steered robot" and "the robot must also use less energy to complete the course than the skid-steered robot." A test course was created and a team of robot experienced and non-robot experienced drivers volunteered to test the maneuverability of EASP and FIRST Team 190's robot, 2k11, from the 2011 season. 2k11 was chosen because it was a 4-wheel tank drive with idling omni-wheels in the rear of the chassis. Also, a team member in the project group had experience driving 2k11 and felt more comfortable with overseeing 2k11's operations than FIRST Team 190's 2014 robot. Since only 2k14 is used for Team 190's demonstrations, 2k11 was the next readily available skid-steered robot.

5.3.1. Obstacle Course

The obstacle course (Figure 27) was designed to test low speed maneuverability and high speed stability of both the EASP and 2k11 robots. The field was about 15 feet wide and 50 feet long with an offshoot area of about 15 feet wide and 20 feet long. Drivers would start at the start line and weave the robot along a slalom of six cones. Cones would be farther apart as robots approached the end of the field. Another cone was placed about seven feet from the end of the field where robots have to loop around. Then, drivers would navigate the robots around an equilateral triangle with sides five feet long. Then, they would drive into an open, 5 foot square box where they have to U-turn around and follow the same path back to the finish line. Hitting a cone was a 20 second penalty.

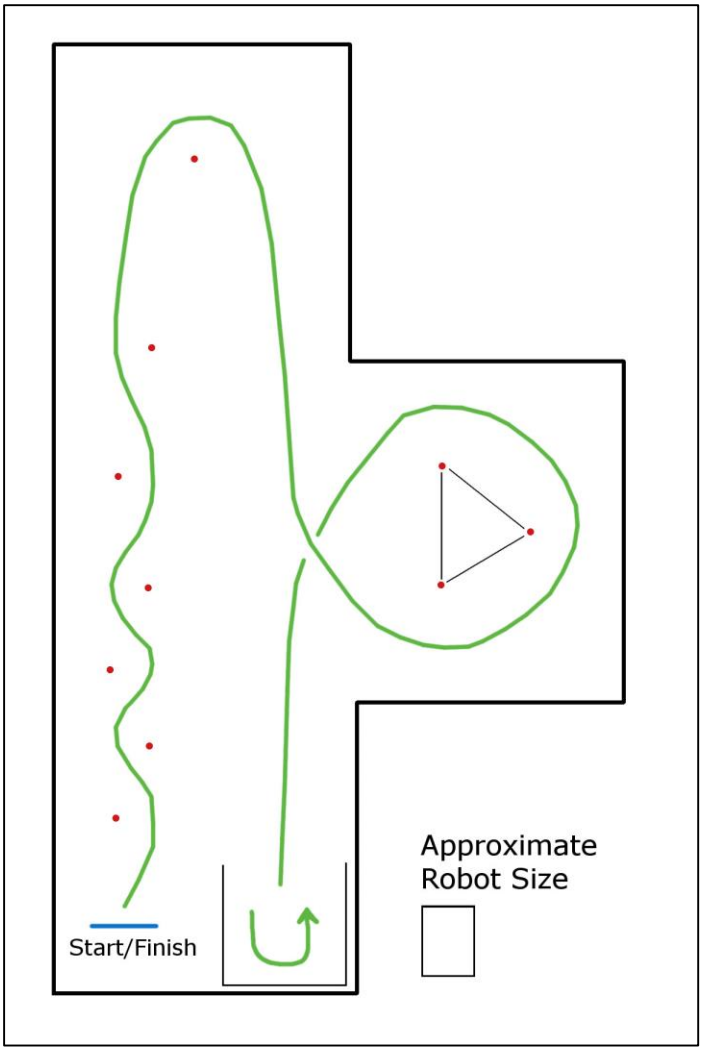


Figure 27: Obstacle Course for Driver Trials (not to scale)

5.3.2. Testing Procedure

Each driver drove the robot around the obstacle twice to see the improvement and learning from each trial. Drivers had one minute to practice driving the robot before driving it around the obstacle course. They would walk along the path of the course before driving the robot. In order to test energy efficiency, the team took the voltage measurement of the battery before and after each trial to observe the voltage drop. The team also recorded the

number of obstacles the robot hit and the completion time. The same process would repeat for the other robot.

5.3.3. Turning Energy Usage

In addition to recording the energy usage based on battery voltage, an ammeter was used to record the current drawn from the battery while the robot was turning in place. The ammeter was hooked onto the battery and its data was recorded when the EASP and 2k11 robots were spinning at 20 revolutions per minute and 60 revolutions per minute.

6. Results and Analysis

6.1. Speed Test

After looking back at the video footage, the team determined that the robot's top speed is 12 feet per second. This result met the project goal that the robot should travel with a speed of at least 10 feet per second.

6.2. Circle Test

Due to the size constraints of the carpet and testing area, the circle was reduced to an eight-foot radius circle with a four-foot wide lane. After attempting the drive EASP along the circle, the team concluded that it is possible to navigate the robot around the circle while keeping it within the four-foot lane at low speeds. It took approximately seven seconds for the robot, at its highest "controllable" speed, to travel one full circle. It was determined that the robot had a speed of 7.2 feet per second. However, at high speeds, the robot was not able to maintain driving within a four-foot lane. If the size of the circle was increased to a ten-foot radius as stated in the project goals, it might have been easier to stay within the circle.

6.3. Skid-Steered Comparison

A total of six robot experienced drivers and seven non-robot experienced drivers drove the EASP and 2k11 robots around the obstacle course.

6.3.1. Obstacle Course Completion Times

On average, all drivers completed the course with EASP in 3 minutes and 55 seconds with approximately five hit obstacles. Therefore, the weighted completion time was 5 minutes and 25 seconds. With 2k11, the average completion time was 4 minutes and 12

seconds, hitting approximately three obstacles. Therefore, the weighted completion was is 5 minutes and 10 seconds.

The robot experienced drivers took, on average, 3 minutes and 31 seconds driving EASP and hitting approximately four obstacles. The weighted completion time was 4 minutes and 57 seconds. With 2k11, they took, on average, 3 minutes and 10 seconds to complete the obstacle course and hit about three obstacles. Therefore, the weighted completion time was 4 minutes and 5 seconds. It would make sense that those with experience driving a tank drive robot would be faster completing the course with 2k11.

For non-robot experienced drivers, the average time completed with EASP was 4 minute and 16 seconds, hitting approximately 5 obstacles. The weighted completion time was 5 minutes and 49 seconds. When they drove 2k11, the average completion time was 5 minutes and 5 seconds and they hit three obstacles. The weighted completion time was 6 minutes and 5 seconds. Therefore, the non-robot experienced drivers had an easier time driving EASP rather than the tank drive robot.

Table 4 shows the average results of all, robot experienced, and non-robot experienced drivers, the robots driven, and the weighted completion time.

Table 4: Average Driver Trial Results

Drivers	Robot	Weighted Completion Time
All	EASP	5:25
	2k11	5:10
Robot Experienced	EASP	4:57
	2k11	4:05
Non-robot Experienced	EASP	5:49
	2k11	6:05

Results for all individual trials can be found in Appendix C: Driver Trial Results.

6.3.2. Voltage Drop

From the results of battery voltage data collected during the trials, it was found that with EASP, the average initial voltage was 12.75 and the final voltage was 12.47, thus giving a voltage drop of 0.28. However, with 2k11, the average initial voltage was 12.99 and the final voltage was 12.57, giving a difference of 0.43. From this data (Table 5), one can see that EASP is more energy efficient than 2k11.

Table 5: Average Voltage Drop for each Robot

Robot	Voltage Drop (V)
EASP	0.28
2k11	0.43

Results for voltage differences from all individual trials can be found in Appendix C: Driver Trial Results.

6.3.3. Turning Energy Usage

In order to determine the power consumption of the robots when turning in place, the team looked at the current data from the battery. It was found that EASP turned about 1/3 revolution per second with about 10 amps drawn from the battery. When turning 1/2 revolution per second, the ammeter read 22 amps and when turning 1 revolution per second, it read 37 amps. When 2k11 turned 1/3 revolution per second, the ammeter read 106 amps. After converting to revolutions per minute and power, the results are shown in Table 6: Turning Energy Usage. EASP used 1/10 power than 2k11 did when turning 20 rpm, making EASP much more energy efficient than 2k11 when turning.

Table 6: Turning Energy Usage

Robot	Revolutions per minute (Approximate)	Energy Usage (Watts)
2k11	20	1256.25
EASP	20	125
	60	462.5

6.4. Achieved Goals

Of the eleven set project goals, eight were met. The robot was teleoperated. It was able to perform a zero radius turn. The error of the steering angles compared to the perfect Ackermann angle did not exceed plus or minus five degrees. The robot travels at least ten

feet per second; the top speed is 12 feet per second. The robot uses significantly less energy than the skid-steered robot to complete the obstacle course. And the robot is compliant to the 2014 FRC rules.

6.5. Unmet Goals

Three of the eleven project goals were not completed. The robot was not able to drive along a ten-foot radius circle while staying within a four-foot wide lane at a speed of ten feet per second. The robot, however, was able to drive along an eight-foot radius circle while staying within a four-foot wide lane at a top speed of 7.2 feet per second. If the robot had traveled any faster, the controller would run into sensitivity issues and it would be much harder to control the robot.

On average of all of the drivers, 2k11 had a faster weighted completion time than EASP by fifteen seconds. Therefore, the goal that, "the robot must complete a course that tests its maneuverability in a time that is on average shorter than the completion time of a 2014 FRC rules-compliant skid-steered robot," was not met. However, when looking at the results of the non-robot experienced drivers, EASP had a faster weighted completion time than 2k11 by sixteen seconds.

The goal that, "the robot must have 250 square inches of continuous space, from the top to the bottom that is by the steering mechanism," was also not met. The wooden boards that were used for mounting the steering mechanism and electronics were also used as a support structure for the frame. Therefore, if the boards were cut to make room for other mechanisms, it would compromise the structure of the chassis.

7. Recommendations

7.1. RC Controller Sensitivity

One of the issues the robot drivers experienced was sensitivity in EASP's RC controller. The controller's steering wheel's turning angle is approximately 90 degrees, while EASP's full turning angle is 224 degrees, as shown in Figure 28. To decrease the sensitivity issue, a controller would be needed where its steering wheel's turning angle would match or exceed that of EASP's. Therefore, a controller with a steering wheel that can turn 224 degrees would alleviate the issue.

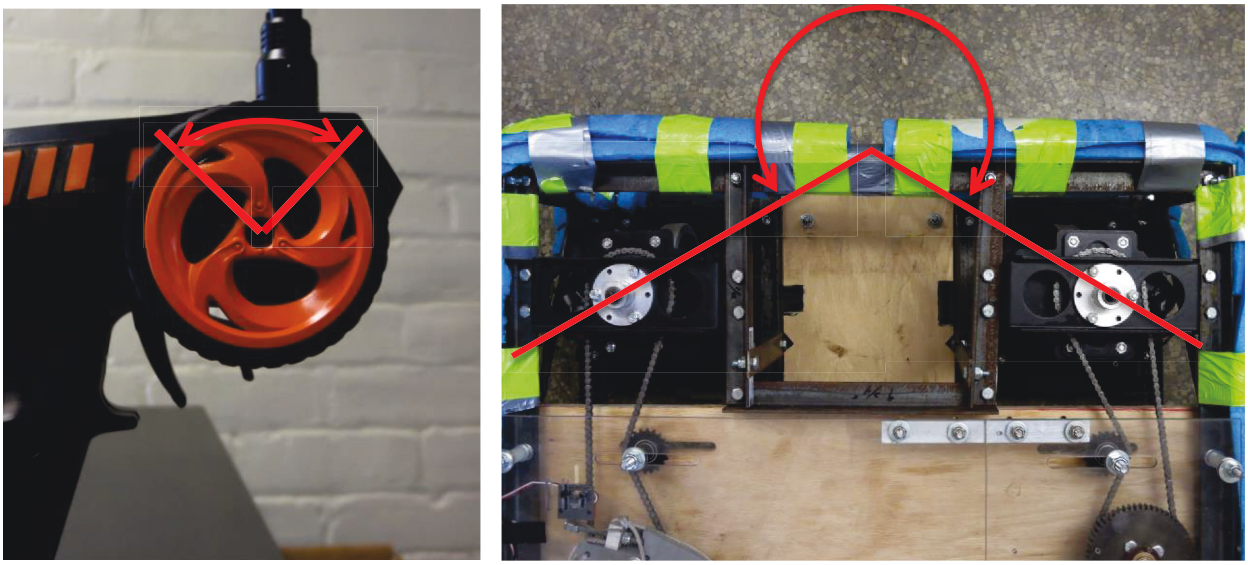


Figure 28: RC Controller's Steering Angle vs. EASP's Steering Angle

7.2. Direction Indicators

Another issue driver's experienced was that they were not able to see how far they were turning. In FRC matches, drivers operate their robots from either end of the field. If one were to operate EASP in a FRC match, it would be difficult for them to know how much the swerve modules were turning without seeing them. From a comment by a robot driver, it

was suggested that there should be direction indicators on the swerve modules (Figure 29) so drivers from a distance could see how far they were turning.

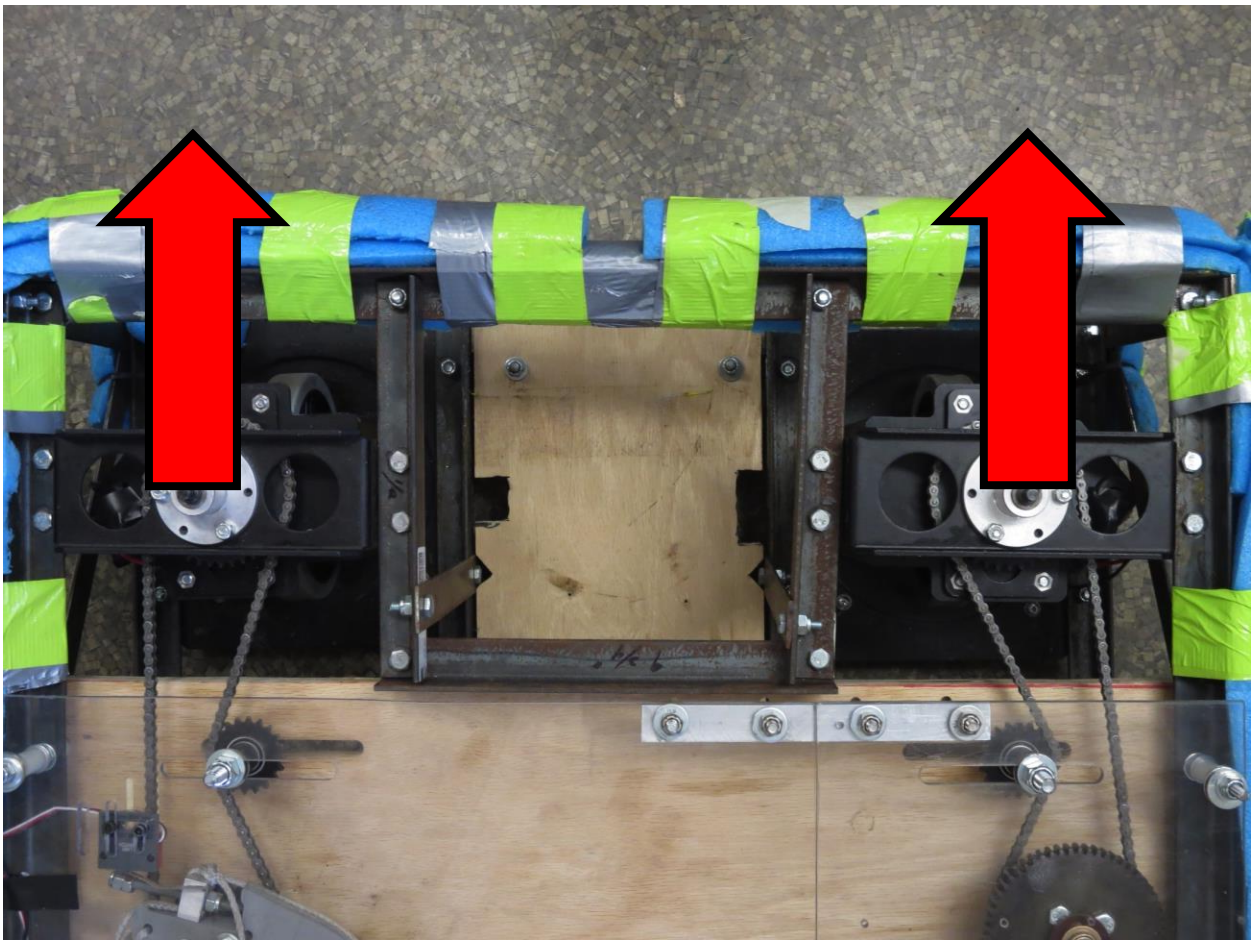


Figure 29: Direction Indicators on Swerve Modules

7.3. Hard stops

When rapidly turning to extremes, the cam-pulleys would slam into the limit switches. To avoid damage to the limit switches, the team had to slow down the PID response. A good improvement would be to put strong hard stops on the limit switches, therefore, the hard stops could take the brunt of the force. This would allow for a faster PID response without having to worry about damage to the limit switches.

8. Social Implications

The EASP design would be beneficial to a number of applications requiring low speed maneuverability and high speed stability, especially in cars and construction equipment. If the EASP design could be implemented in cars, it would simplify parallel parking. It would also allow the vehicle to fit into tight parking spaces. This could decrease the number of parking accidents.

Construction equipment could benefit from the EASP design. Construction equipment used in the city will need low speed maneuverability, especially on smaller properties. One of the disadvantages of farming equipment was the slow speed. Therefore, if EASP's design was integrated in farming equipment, it would potentially increase the speed while still maintaining tight turns. It would also greatly increase the energy efficiency of the equipment. Also, treaded construction vehicles can tear up the ground when turning. This would be less a problem if our system was implemented.

9. Conclusion

In conclusion, the EASP is much more maneuverable than a standard Ackermann system. The non-experienced drivers performed better with EASP than 2k11. However, the experienced drivers performed better with 2k11 than EASP. This is due to the experienced drivers' prior ability to operate a tank drive robot and are therefore, more comfortable to operating a skid-steered robot. Because the RC controller is meant for hobby cars and our robot drives similar to a car, driving EASP became very intuitive and the difference in completion time between 2k11 and EASP and the experienced and non-experienced drivers was very small.

EASP was also significantly more energy efficient than 2k11, especially when turning. Based on the current test, 2k11 used ten times more power than EASP when turning 20 rpm. The measurement of voltage drop as a means to determine energy efficiency was too unreliable to draw a quantitative conclusion about efficiency when completing the obstacle course. However based on qualitative observation, the 2k11 robot ran out of battery faster than EASP. Overall, the robot met the goal of being more energy efficient than a skid-steered FRC robot.

In conclusion, the team met all but three goals. If recommendations were implemented to the robot, it would be a very maneuverable robot in FRC competitions and would be beneficial in cars and construction equipment.

References

[1] "Wheel Control Theory," [Online]. Available: http://www.robotplatform.com/knowledge/Classification_of_Robots/wheel_control_theory.html. [Accessed 5 October 2014].

[2] R. N. Jazar, Vehicle Dynamics: Theory and Applications, New York: Springer Science+Business Media, 2009.

[3] S. D. W. D. K. G. Michael Cullen, "OPTIMAL DRIVELINE ROBOT BASE," 1 May 2014. [Online]. Available: http://www.wpi.edu/Pubs/E-project/Available/E-project-043014-014300/unrestricted/01_ODRB_Final_Paper.pdf. [Accessed 5 October 2014].

[4] J.-S. Z. Z.-J. F. Sheng Zhao, "Design of a pair of noncircular gears meeting Ackermann steering principle," in *Consumer Electronics, Communications and Networks (CECNet), 2011 International Conference on* , XianNing, 2011.

[5] "Non-circular gear," [Online]. Available: http://en.wikipedia.org/wiki/Non-circular_gear#mediaviewer/File:Non-circular_gear.PNG. [Accessed 5 Octorber 2014].

[6] "PS3 Racing Wheel Controller," [Online]. Available: http://www.amazon.com/PS3-Racing-Wheel-Controller-Playstation-3/dp/B004G5TUJM. [Accessed 5 October 2014].

[7] "Himoto Syclone RC Nitro Buggy 1/10 RTR 4WD (Flame 2 Speed 60mph)," [Online]. Available: http://www.rcxmodels.com/rc-car-nitro-buggy. [Accessed 5 October 2014].

[8] "Beta-2010-electronics," [Online]. Available: http://files.andymark.com/Beta-2010-electronics.jpg. [Accessed 5 October 2014].

[9] FRC Team 190, "2014 FRC Motors," 2014.

[10] "T6AP 6CH RC for airplane 17cm*8.5cm*20cm," [Online]. Available: http://www.cnbestone.com/index.php?gOo=goods_details.dwt&goodsid=603&productname=. [Accessed 5 October 2014].

[11] "Radio Control Systems and How they work," [Online]. Available: http://www.rc-airplane-advisor.com/radio-control-systems.html. [Accessed 5 October 2014].

[12] S. Y. Nof, Handbook of Industrial Robotics, New York: John Wiley & Sons, 1999.

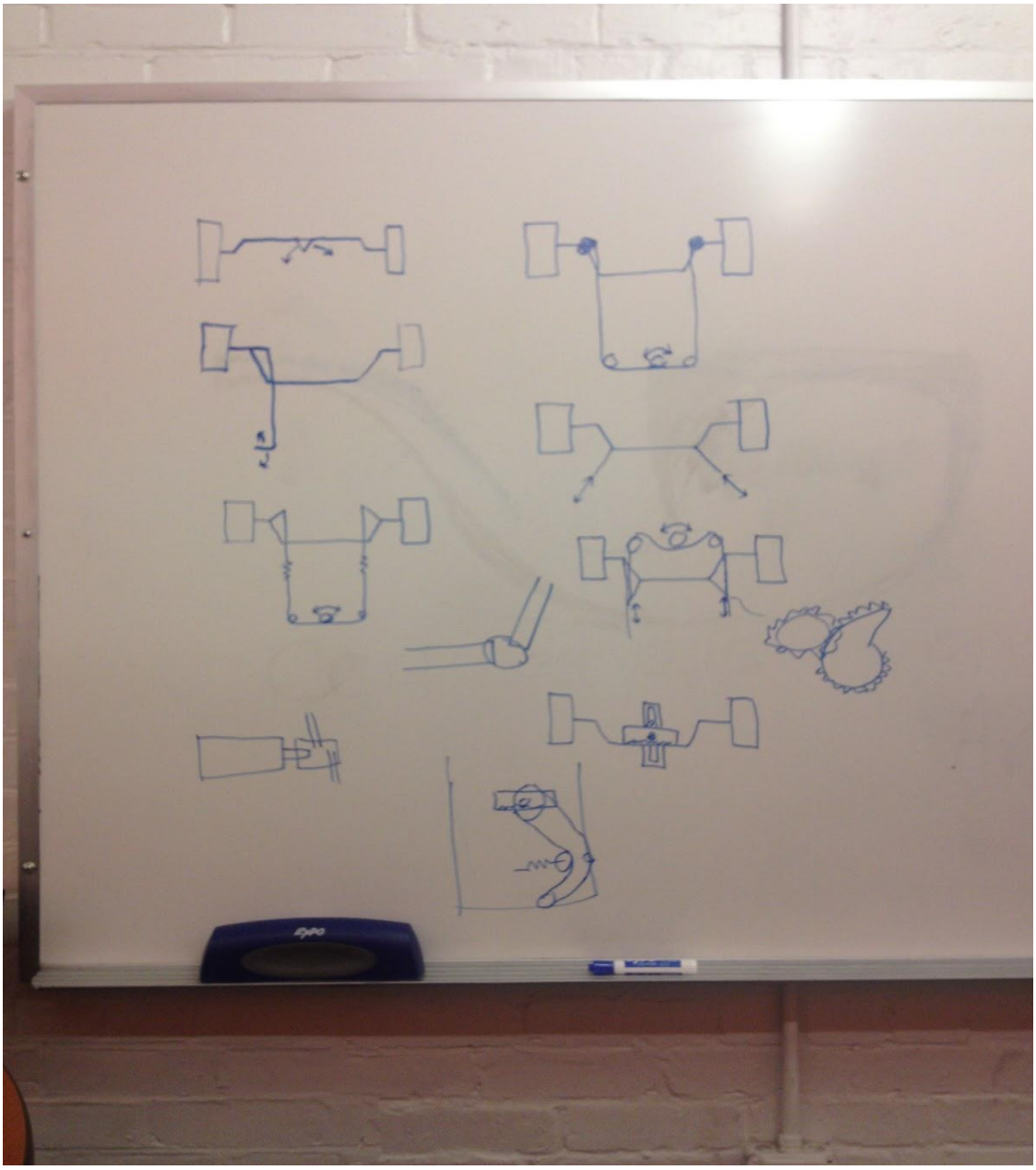
[13] "TELEOPERATED ROBOTS," Consortium on Cognitive Science Instruction, [Online]. Available: http://www.mind.ilstu.edu/curriculum/medical_robotics/teleo.php. [Accessed 5 October 2014].

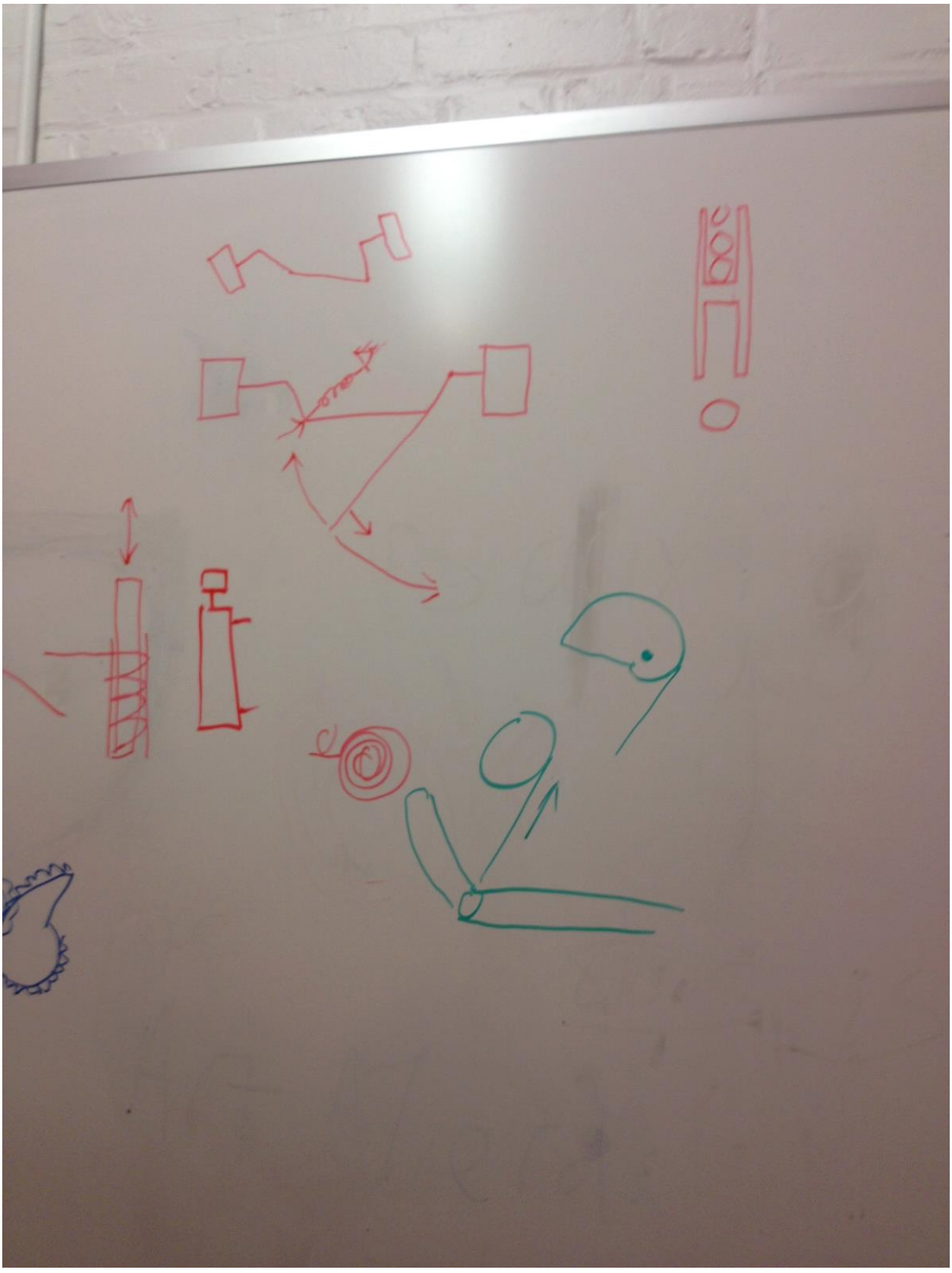
[14] "Human Exploration Telerobotics (HET)," NASA, [Online]. Available: http://www.nasa.gov/mission_pages/tdm/telerobotics/telerobotics_overview.html#.VDKUEhbpe2B. [Accessed 5 October 2014].

[15] FIRST, "Game Manual," FIRST, [Online]. Available: http://frc-manual.usfirst.org/viewItem/3. [Accessed 5 October 2014].

Appendices

Appendix A. Brainstorm Sketches





Appendix B: Code

EAS.ino

```
/* EAS.ino (Enhanced Ackermann Steering
 * Will Parker
 *
 *
 */

#include <StandardCplusplus.h>
#include <ServoTimer2.h>
#include <Math.h>
#include <TimerOne.h>
#include <limits.h>
#include "Controller.h"
#include "Steering.h"
#include "Motor.h"
#include "Util.h"

/* print information to serial port for debugging.
 * WARNING, DO NOT POWER ON ROBOT WHEN PRINTING.
 * The print statement slow down the PID loop and
 * the robot will probably crash.
 */
//#define DEBUG

#define DRIVE_PER 100

// kill switches, one for 5V, one for ground
#define POS_KS 8
#define NEG_KS 7

// time elapsed in ms
volatile unsigned long ms = 0;

// time of next PID/velocity update in ms
unsigned long nextSteeringUpdate = 0;
unsigned long nextDriveUpdate = 0;

Steering vanDoor;  // van door steering motor

/* must initialize with dummy or else PWM won't work for every motor.
 * I'm not exactly sure why, but it probably has something to
 * do with initializing to many ServoTimer2 objects.
 */
Motor dummy;

Motor drives[4] = {
  dummy, dummy, dummy, dummy};

void setup() {
  vanDoor = Steering(STEERING, SERVO_MIN, SERVO_MAX, STEERING_POT);
  /* toggle steering motor direction.
   * BE CAREFULL WITH THIS. If the motor is turning the wrong
```

```
   * direction, the steering mechanism will crash
   */
  vanDoor.swapPolarity = false;

  drives[0] = Motor(LF_MOTOR, MOTOR_MIN, MOTOR_MAX);
  drives[1] = Motor(RF_MOTOR, MOTOR_MIN, MOTOR_MAX);
  drives[1].setSwapPolarity(true);
  drives[2] = Motor(LR_MOTOR, MOTOR_MIN, MOTOR_MAX);
  drives[3] = Motor(RR_MOTOR, MOTOR_MIN, MOTOR_MAX);
  drives[2].setSwapPolarity(true);

  // set locations of drive wheels
  drives[0].setPos(-WHEEL_TRACK/2, WHEEL_BASE);
  drives[1].setPos(WHEEL_TRACK/2, WHEEL_BASE);
  drives[2].setPos(-WHEEL_TRACK/2, 0);
  drives[3].setPos(WHEEL_TRACK/2, 0);

  // pin mode for controller inputs
  pinMode(THROTTLE, INPUT);
  pinMode(STEERING_WHEEL, INPUT);

  // limit switches
  pinMode(LEFT_LIMIT, INPUT_PULLUP);
  pinMode(RIGHT_LIMIT, INPUT_PULLUP);

  // kill switches, POS_KS should be connected to an external pulldown
resistor
  pinMode(POS_KS, INPUT);
  pinMode(NEG_KS, INPUT_PULLUP);

  // start timer so that it ticks every 1000 microseconds (1 millisecond)
  Timer1.initialize(1000);
  Timer1.attachInterrupt(tick);

# ifdef DEBUG
  Serial.begin(9600);
# endif
}

void loop() {
  ControlInput control = getControlInputs();
  checkKS();  // check kill switches

  // steering
  if (ms >= nextSteeringUpdate) {
    vanDoor.updatePos(control.steeringWheel);
    nextSteeringUpdate = ms + STEERING_PER;

# ifdef DEBUG
    steeringReadOut(control);
# endif

  }

  // throttle
  if (ms >= nextDriveUpdate) {
    // calculate distances from each wheel to the turning center
```

```
    int maxTurnDist = calcTurnDistances();

    for (int i = 0; i < 4; i++) {
      drives[i].calcVel(control.throttle, maxTurnDist);
      drives[i].updateVelocity();
    }
    nextDriveUpdate = ms + DRIVE_PER;

# ifdef DEBUG
    throttleReadOut(control);
# endif
  }
}

void tick() {
  ms++;
}

// calc the x position of the turning center
int calcTurningCenter(Steering steering) {
  int x;  // the turning center pos

  double rightAng = steering.rightAngle();

  if (rightAng == 0) {  // straight
    x = INT_MAX;  // largest possible int
  }
  else {
    x = WHEEL_TRACK / tan(-rightAng) + WHEEL_BASE / 2;
  }
  return x;
}

// calculate the turn distances for all of the motors and return the maximum
int calcTurnDistances() {
  int turnCenter = calcTurningCenter(vanDoor);
  int dir = vanDoor.getDirection();
  int maxTurnDist = 0;

  for (int i = 0; i < 4; i++) {
    drives[i].calcTurnDist(turnCenter, dir);
    if (drives[i].turnDist > maxTurnDist) {
      maxTurnDist = drives[i].turnDist;
    }
  }

  return maxTurnDist;
}


void steeringReadOut(ControlInput control) {
  Serial.println("|--------------------------- Steering Read Out ------------
---------------|");

  Serial.print("Raw Steering Input: ");
  Serial.println(pulseIn(STEERING_WHEEL, HIGH, CNTL_TMOUT));
```

```
  Serial.print("Mapped Steering Input: ");
  Serial.println(radToDeg(control.steeringWheel));

  Serial.print("Raw Potentiometer Value: ");
  Serial.println(vanDoor.readPos());

  Serial.print("Current Direction (Based on Pot Val): ");
  int dir = vanDoor.getDirection();
  if (dir > 0) Serial.println("Left");
  else if (dir < 0) Serial.println("Right");
  else Serial.println("Straight");

  Serial.print("Van Door Motor Angle (Radians, Degrees): ");
  double radAng = vanDoor.getAngle();
  Serial.print(radAng, 3);
  Serial.print(", ");
  Serial.println(radToDeg(radAng), 1);

  Serial.print("Left Wheel Angle (Radians, Degrees): ");
  double leftRadAng = vanDoor.leftAngle();
  Serial.print(leftRadAng, 3);
  Serial.print(", ");
  Serial.println(radToDeg(leftRadAng), 1);

  Serial.print("Right Wheel Angle (Radians, Degrees): ");
  double rightRadAng = vanDoor.rightAngle();
  Serial.print(rightRadAng, 3);
  Serial.print(", ");
  Serial.println((rightRadAng * 180)/PI, 1);

  Serial.print("Left Limit Switch: ");
  if (digitalRead(LEFT_LIMIT)) Serial.println("High");
  else Serial.println("Low");

  Serial.print("Right Limit Switch: ");
  if (digitalRead(RIGHT_LIMIT)) Serial.println("High");
  else Serial.println("Low");

  Serial.println("|------------------------- End Steering Read Out ----------
--------------|\n");
}

void throttleReadOut(ControlInput control) {
  Serial.println("|------------------------- Drive Motor Read Out ----------
--------------|");

  Serial.print("Raw Throttle Input: ");
  Serial.println(pulseIn(THROTTLE, HIGH, CNTL_TMOUT));

  Serial.print("Throttle In: ");
  Serial.println(control.throttle);

  Serial.print("Turning Center: ");
  int tc = calcTurningCenter(vanDoor);
  Serial.println(tc);

  Serial.println("Front Left Motor: ");
```

```
  Serial.print("  Turning Radius: ");
  Serial.println(drives[0].turnDist);
  Serial.print("  Velocity: ");
  Serial.println(drives[0].vel);

  Serial.println("Front Right Motor: ");
  Serial.print("  Turning Radius: ");
  Serial.println(drives[1].turnDist);
  Serial.print("  Velocity: ");
  Serial.println(drives[1].vel);

  Serial.println("Front Left Motor: ");
  Serial.print("  Turning Radius: ");
  Serial.println(drives[2].turnDist);
  Serial.print("  Velocity: ");
  Serial.println(drives[2].vel);

  Serial.println("Front Left Motor: ");
  Serial.print("  Turning Radius: ");
  Serial.println(drives[4].turnDist);
  Serial.print("  Velocity: ");
  Serial.println(drives[4].vel);

  Serial.println("|------------------------ End Drive Motor Read Out --------
---------------|\n");
}

void checkKS() {
  if ((digitalRead(POS_KS) == LOW) || (digitalRead(NEG_KS) == HIGH)) {
    wdtReset();
  }
}
```

Controller.cpp

```
/* Controller.cpp
 * Will Parker
 */

#include "Controller.h"
#include "Steering.h"
#include "Motor.h"
#include "Util.h"
#include "arduino.h"

ControlInput getControlInputs() {
  ControlInput control;
  control = getRawInputs();
  control = mapControls(control);
  return control;
}

ControlInput getRawInputs() {
  ControlInput control;
  control.throttle = pulseIn(THROTTLE, HIGH, CNTL_TMOUT);
  control.steeringWheel = pulseIn(STEERING_WHEEL, HIGH, CNTL_TMOUT);
```

```
  return control;
}

ControlInput mapControls(ControlInput control) {
  // if out of range
  if ((control.throttle > THROTTLE_MAX + 200) || (control.throttle <
THROTTLE_MIN - 200)
    || (control.steeringWheel > STEERING_WHEEL_MAX + 200) ||
(control.steeringWheel < STEERING_WHEEL_MIN - 200)) {
    control.throttle = 0;  // stop
    control.steeringWheel = 0;
    wdtReset();
  }
  else {
    control.throttle = map(control.throttle, THROTTLE_MIN, THROTTLE_MAX, -
500, 500);
    if (control.steeringWheel > STEERING_WHEEL_STRAIGHT +
STEERING_FINE_OFFSET) {
      control.steeringWheel =
        doubleMap(control.steeringWheel, STEERING_WHEEL_STRAIGHT +
STEERING_FINE_OFFSET, STEERING_WHEEL_MAX,
      FINE_ANGLE_OFFSET, MAX_INNER_ANGLE*1.1);
    }
    else if (control.steeringWheel < STEERING_WHEEL_STRAIGHT -
STEERING_FINE_OFFSET) {
      control.steeringWheel =
        doubleMap(control.steeringWheel, STEERING_WHEEL_MIN,
STEERING_WHEEL_STRAIGHT - STEERING_FINE_OFFSET,
      -MAX_INNER_ANGLE*1.1, -FINE_ANGLE_OFFSET);
    }
    else {
      control.steeringWheel =
        doubleMap(control.steeringWheel, STEERING_WHEEL_STRAIGHT -
STEERING_FINE_OFFSET, STEERING_WHEEL_STRAIGHT + STEERING_FINE_OFFSET,
      -FINE_ANGLE_OFFSET, FINE_ANGLE_OFFSET);
    }
  }

  return control;
}
```

Controller.h

```
/* Controller.h
 * Will Parker
 */

#ifndef Controller_h
#define Controller_h

// controller pins
#define THROTTLE 12
#define STEERING_WHEEL 13

// timeout period for controller pulses in microseconds
#define CNTL_TMOUT 25000
```

```
// minimum and maximum raw rc input values
#define THROTTLE_MIN 900
#define THROTTLE_MAX 1829
#define STEERING_WHEEL_MIN 930
#define STEERING_WHEEL_MAX 1864
#define STEERING_WHEEL_STRAIGHT 1400
#define STEERING_FINE_OFFSET 200

// struct containing the throttle and steering inputs
typedef struct ControlInput {
  int throttle;
  double steeringWheel;
}ControlInput;

ControlInput getControlInputs();

ControlInput getRawInputs();

ControlInput mapControls(ControlInput control);

#endif
```

Motor.cpp

```
/* Motor.cpp
 * Will Parker
 */

#include "Motor.h"
#include "Util.h"

Motor::Motor(){
}

Motor::Motor(int pin, int minWrite, int maxWrite) :
minWrite(minWrite), maxWrite(maxWrite) {
  motor.attach(pin, minWrite, maxWrite);
  swapPolarity = false;
  turnDist = INT_MAX;
  vel = MOTOR_STOP;
}

void Motor::updateVelocity() {
  int writeVel = vel;
  if (swapPolarity) writeVel = (MOTOR_MAX - MOTOR_MIN) - (vel - MOTOR_MIN) +
MOTOR_MIN;
  motor.write(writeVel);
}

void Motor::setSwapPolarity(boolean _swapPolarity) {
  swapPolarity = _swapPolarity;
}

void Motor::fullStop() {
  motor.write(MOTOR_STOP);
```

```
}

void Motor::setPos(int x, int y) {
  xPos = x;
  yPos = y;
}

void Motor::calcTurnDist(int turnCenter, int dir) {
  turnDist = linearDist(xPos, yPos, turnCenter, 0);

  if (yPos == 0) {  // non steered wheel
    if ((dir > 0 && turnCenter > xPos)
      || (dir < 0 && turnCenter < xPos)) {
      turnDist = -turnDist;
    }
  }
}

void Motor::calcVel(int velIn, int maxTurnRadius) {
  vel = velIn * ((float)turnDist / maxTurnRadius) + MOTOR_STOP;
}
```

Motor.h

```
/* Motor.h
 * Will Parker
 */

#ifndef Motor_h
#define Motor_h

#include "arduino.h"
#include <ServoTimer2.h>
#include "Pid.h"
#include "Util.h"

// motor pins
#define LF_MOTOR 11
#define RF_MOTOR 6
#define LR_MOTOR 9
#define RR_MOTOR 10

// min and max motor controller write values
#define MOTOR_MIN 1000
#define MOTOR_MAX 2000
#define MOTOR_STOP ((MOTOR_MAX - MOTOR_MIN) / 2 + MOTOR_MIN)

class Motor {
  ServoTimer2 motor;
  int minWrite, maxWrite;
  boolean swapPolarity; // used to swap polarity

public:
  // posistion of drive wheel relative to virtual turning center
  int xPos, yPos;
  int turnDist, vel;
```

```
  Motor();
  Motor(int pin, int minWrite, int maxWrite);
  void updateVelocity();
  void setSwapPolarity(boolean _swapPolarity);
  void fullStop();
  void setPos(int x, int y);
  void calcTurnDist(int turnCenter, int dir);
  void calcVel(int velIn, int maxTurnRadius);
};

void updateDrive(int throttle, int steeringPos);

#endif
```

Pid.cpp

```
/* Pid.cpp
 * Will Parker
 */

#include "Pid.h"
#include "arduino.h"
#include "util.h"

Pid::Pid(){
}

Pid::Pid(int p, int i, int d, double maxTotal) :
p(p), i(i), d(d), maxTotal(maxTotal) {
}

int Pid::calcOutput(double curVal, double desVal) {
  double error = curVal - desVal;

  // keep totalError within the bounds of the maxTotal
  totalError = min(totalError, maxTotal);
  totalError = max(totalError, -maxTotal);

  double errorChange = error - prevError;

  if (abs(errorChange) < 0.01) {
    totalError += error;
  }

  if ((totalError > 0 && error < 0) || (totalError < 0 && error > 0))
totalError = 0;

  int output = p * error + i * totalError + d * errorChange;

  prevError = error;  // save error for next update

  return output;
}
```

Pid.h

```
/* Pid.h
 * Will Parker
 */

#ifndef Pid_h
#define Pid_h

// period of PID updates in ms
#define STEERING_PER 20

class Pid {
  int p, i, d;
  double totalError, prevError, maxTotal;

public:
  Pid();
  Pid(int p, int i, int d, double maxTotal);
  int calcOutput(double curVal, double desVal);
};

#endif
```

Steering.cpp

```
/* Steering.cpp
 * Will Parker
 */

#include "Steering.h"
#include "Pid.h"
#include "Util.h"
#include <math.h>

Steering::Steering() {
}

Steering::Steering(int pin, int minWrite, int maxWrite, int pot) :
minWrite(minWrite), maxWrite(maxWrite), pot(pot) {
  motor.attach(pin, minWrite, maxWrite);
  pid = Pid(STEERING_P, STEERING_I, STEERING_D, STEERING_MAX_TOTAL);
  swapPolarity = false;
}

void Steering::updatePos(double desPos) {
//  int potval = readPos();
  double motorAng = getAngle();
  double curPos = 0;
  if (motorAng < 0) {  // turning left
    curPos = leftAngle();
  }
  else {  // turning right
    curPos = rightAngle();
  }
  int posChange = pid.calcOutput(curPos, desPos);
```

```
  if (swapPolarity) posChange = - posChange;
  turn(posChange + SERVO_STOP);
}

// turn motor without pid
void Steering::turn(int vel) {
    if ((hitLeftLimit() && vel > SERVO_STOP)
      || (hitRightLimit() && vel < SERVO_STOP)) {  // if trying to turn past
limit
      motor.write(SERVO_STOP);
    }
    else {
      motor.write(vel);
    }
}

// raw pot position
int Steering::readPos() {
  return analogRead(pot);
}

// positive value for left, negative for right, 0 for straight
int Steering::getDirection() {
  return -(analogRead(pot) - STRAIGHT);
}

// calc the angle of the van door motor
double Steering::getAngle() {
  int potVal = readPos();
  double potAngle = (potVal - STRAIGHT) * POT_RES;

  return potAngle;
}

double Steering::leftAngle() {
  double angle = -getAngle() * MOTOR_RATIO;

  if (angle != 0) {  // if turning
    // ackermann equation
    double ackAng = atan2(1, (1/tan(angle)) - (WHEEL_TRACK/WHEEL_BASE));
    if (angle < 0) {
      ackAng -= PI; // offset if negative
    }
    return ackAng;
  }
  else return 0;  // straight ahead
}

double Steering::rightAngle() {
  return -getAngle() * MOTOR_RATIO;
}

boolean hitLeftLimit() {
  return  digitalRead(LEFT_LIMIT) == LOW;
}

boolean hitRightLimit() {
```

```
  return  digitalRead(RIGHT_LIMIT) == LOW;
}
```

Steering.h

```
/* Steering.h
 * Will Parker
 */

#ifndef Steering_h
#define Steering_h

#include "arduino.h"
#include "Pid.h"
#include <ServoTimer2.h>

#define STEERING 5 // digital I/O pin for steering motor (PWM)

#define STEERING_POT 5 // analog read pot pin

/* digital I/O pins for left and right limit switches
 * (ie. triggered when turning full left or full right)
 */
#define LEFT_LIMIT 2
#define RIGHT_LIMIT 4

// min and max servo write values
#define SERVO_MIN 600
#define SERVO_MAX 2400
#define SERVO_STOP (SERVO_MAX - SERVO_MIN) / 2 + SERVO_MIN

// min and max pot values
#define STEERING_POT_MIN 0
#define STEERING_POT_MAX 1023

#define POT_ANGLE_RANGE ((270/180.0)*PI)

// min and max pot angle in radians
#define POT_RES POT_ANGLE_RANGE/STEERING_POT_MAX
//#define STEERING_POT_ANGLE_MIN (-100.0/180.0)*PI
//#define STEERING_POT_ANGLE_MAX (100.0/180.0)*PI
#define MAX_INNER_ANGLE ((115.0/180.0)*PI)
#define FINE_ANGLE_OFFSET ((10.0/180.0)*PI)

#define POT_OFFSET 112

#define STRAIGHT (STEERING_POT_MAX/2 - POT_OFFSET)  // pot value for straight
ahead

// ratio of steering motor angle to output angle
#define MOTOR_RATIO 0.75

// wheel base and track in inches
# define WHEEL_TRACK 17
# define WHEEL_BASE 20.5
```

```
// PID VALUES
#define STEERING_P 150.0
#define STEERING_I (130.0 * (STEERING_PER / 20.0))
#define STEERING_D 100.0
#define STEERING_MAX_TOTAL 15 // maximum total PID error



class Steering {
  ServoTimer2 motor;
  Pid pid;
  int pot, minWrite, maxWrite;

public:
  boolean swapPolarity; // used to swap polarity

  Steering();
  Steering(int pin, int minWrite, int maxWrite, int pot);
  void updatePos(double desPos);
  void turn(int vel);
  int getDirection();
  int readPos();
  double getAngle();
  double leftAngle();
  double rightAngle();
};

boolean hitLeftLimit();
boolean hitRightLimit();

#endif
```

Util.cpp

```
/* Util.cpp
 * Will Parker
 */

#include "Util.h"
#include "arduino.h"
#include <avr/wdt.h>

double doubleMap(double input, double fromMin, double fromMax, double toMin,
double toMax) {
  return (input - fromMin) * (toMax - toMin) / (fromMax - fromMin) + toMin;
}

int linearDist (int x1, int y1, int x2, int y2) {
  return sqrt(pow(x1-x2, 2) + pow(y1-y2, 2));
}

int radToDeg(float rad) {
 return rad*180/PI;
}

void wdtReset() {
```

```
  wdt_enable(WDTO_15MS);
  while(1)
  {
  }
}
```

Util.h

```
/* Util.h
 * Will Parker
 */

#ifndef Util_h
#define Util_h

#define INT_MAX 32765

extern volatile unsigned long ms; // number of elapsed milliseconds

double doubleMap(double input, double fromMin, double fromMax, double toMin,
double toMax);

int linearDist(int x1, int y1, int x2, int y2);

int radToDeg(float rad);

void wdtReset();

#endif
```

Appendix C: Driver Trial Results

Driver	FRC Experience	Robot	Completion Time	Obstacles Hit	Adjusted Completion Time	Battery Name	Initial Voltage (V)	Final Voltage (V)	Voltage Drop (V)
Driver 1	Yes	2k11	02:19	7	04:39	Lawrence	12.71	12.45	0.26
		2k11	02:49	4	04:09	Bonnie	13.06	12.70	0.36
		EAS	03:57	9	06:57	Bonnie	12.55	12.3	0.25
		EAS	03:30	9	06:30	Bonnie	12.3	12.14	0.16
Driver 2	No	EAS	01:52	5	03:32	Phyllis	12.9	12.77	0.13
		EAS	01:44	5	03:24	Phyllis	12.77	12.57	0.2
		2k11	02:21	4	03:41	Lawrence	13.41	12.8	0.61
		2k11	02:26	1	02:46	Bonnie	13.05	12.72	0.33
Driver 3	No	2k11	05:54	7	08:14	Don't	13.07	12.57	0.5
		2k11	05:26	3	06:26	Larry	12.87	12.22	0.65
		EAS	04:48	7	07:08	Lawrence	12.59	12.4	0.19
		EAS	04:19	3	05:19	Phyllis	12.9	12.39	0.51
Driver 4	Yes	EAS	05:55	3	06:55	Lawrence/ Bonnie	12.33, 13.08	12.21,12.75	0.12, 0.33
		EAS	05:00	2	05:40	Bonnie	12.75	12.61	0.14
		2k11	04:21	2	05:01	Phyllis	13.06	12.71	0.35
		2k11	03:41	2	04:21	Phyllis	12.71	12.4	0.31
Driver 5	Yes	2k11	05:04	4	06:24	Don't	13	12.42	0.58
		2k11	02:50	8	05:30	Don't	13.03	12.39	0.64
		EAS	04:26	12	08:26	Bonnie	12.7	12.35	0.35
		EAS	03:21	8	06:01	Lawrence/ Bonnie	12.75, 12.67	12.32,12.31	0.43, 0.36
Driver 6	Yes	EAS	03:01	2	03:41	Lawrence/ Bonnie	12.70, 12.65	12.27, 12.32	0.43, 0.35
		EAS	03:01	0	03:01	Phyllis	12.82	12.33	0.49
		2k11	02:46	3	03:46	Lawrence	12.91	12.66	0.25
		2k11	03:12	0	03:12	Larry	13.06	12.61	0.45
Driver 7	Yes	2k11	02:19	0	02:19	Beverly	13.14	12.67	0.47
		2k11	01:59	1	02:19	Lawrence	13.07	12.63	0.44
		EAS	02:02	3	03:02	Don't	12.57	12.05	0.52
		EAS	01:53	1	02:13	Bonnie	12.69	12.33	0.36
Driver 8	Yes	EAS	03:02	2	03:42	Beverly	12.92	12.7	0.22
		EAS	03:00	1	03:20	Beverly	12.7	12.54	0.16
		2k11	03:37	0	03:37	Phyllis	12.84	12.56	0.28
		2k11	03:02	2	03:42	Don't	13.01	12.66	0.35
Driver 9	No	EAS	03:18	6	05:18	Larry	12.67	12.3	0.37
		EAS	03:14	4	04:34	Beverly	13.05	12.51	0.54
		2k11	05:48	1	06:08	Lawrence	12.9	12.52	0.38

		2k11	05:02	1	05:22	Bonnie/ Lawrence	13.04, 12.77	12.21, 12.47	0.83, 0.30
Driver 10	No	2k11	05:15	8	07:55	Phyllis	12.86	12.35	0.51
		2k11	04:18	3	05:18	Beverly	13.05	12.68	0.37
		EAS	03:17	9	06:17	Phyllis	12.69	12.48	0.21
		EAS	03:45	5	05:25	Larry	12.81	12.29	0.52
Driver 11	No	2k11	07:35	0	07:35	Beverly	13	12.72	0.28
		2k11	07:31	0	07:31	Bonnie	13.06	12.53	0.53
		EAS	07:55	3	08:55	Phyllis	12.86	12.79	0.07
		EAS	08:55	1	09:15	Phyllis	12.79	12.63	0.16
Driver 12	No	EAS	04:35	5	06:15	Lawrence	12.79	12.5	0.29
		EAS	03:37	2	04:17	Don't	12.98	12.85	0.13
		2k11	04:30	3	05:30	Beverly	12.94	12.57	0.37
		2k11	04:58	3	05:58	Larry	13.12	12.57	0.55
Driver 13	No	2k11	05:19	5	06:59	Larry	12.92	12.46	0.46
		2k11	04:45	3	05:45	Beverly	12.94	12.58	0.36
		EAS	04:25	5	06:05	Bonnie	12.83	12.56	0.27
		EAS	04:06	5	05:46	Bonnie	12.56	12.38	0.18
Avg.	All	EAS	03:55	4.5	05:25		12.75	12.47	0.28
		2k11	04:12	2.88461538	05:10		12.99	12.57	0.43
Avg.	No	EAS	04:16	4.64285714	05:49				
		2k11	05:05	3	06:05				
Avg.	Yes	EAS	03:31	4.33333333	04:57				
		2k11	03:10	2.75	04:05				