Griffin Robot Design Document

Project Goal

Build a robot to compete in the First Tech Challenge for 2019-2020, placing well enough to compete in the New Mexico FIRST Tech Challenge Championship in Albuquerque, New Mexico.

Motorized Base

The robot base is a rectangle frame made from 160 mm and 290 mm aluminum C-channel structural elements. The corners are 96 mm C-channel com[onents belo, with a 96 mm flat bracket cover on top. The motors mount to the underside of the 96 mm C-channel corner elements. The arrangement of these components can be seen in the following figure. (See Fig. 1.)

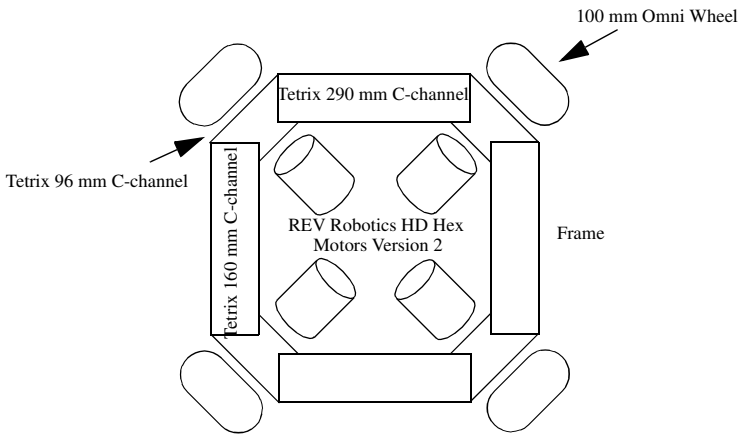

Figure 1 - Robot Base And Drive

Parallel Bar Lift System

The robot has a simple parallel bar lift system that allows the robot to capture and raise game elements. The lift motor is a single 12-volt DC motor that rotates a pair of 24-tooth gears. The rotation of the drive gear causes a second gear to rotate via a chain mechanism that is held tight by a nylon tension wheel. The second gear rotates the arm upwards and downwards. This is shown in Figure 2. This mechanism extends the collection mechanism forward and upward at the same time, holding the game piece even with the ground.

The capture mechanism is a servo with a bar that swings upward and downward 90 degrees. The bar pins the SkyStone or other game piece to the back plate while the robot is in motion.

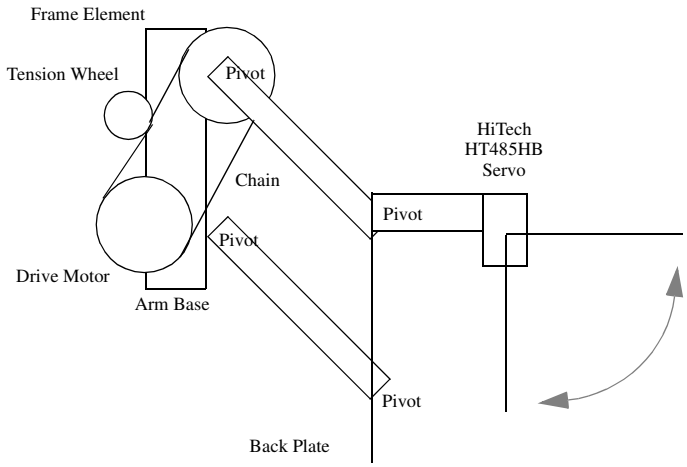

Figure 2 - Scissor Lift System

The hook mechanism mounts below the frame and allows the robot to attach to the foundation game element, to maneuver the foundation as needed. It is powered by a single HiTech HT485HB servo, as shown in the following diagram. (See Fig. 3.)

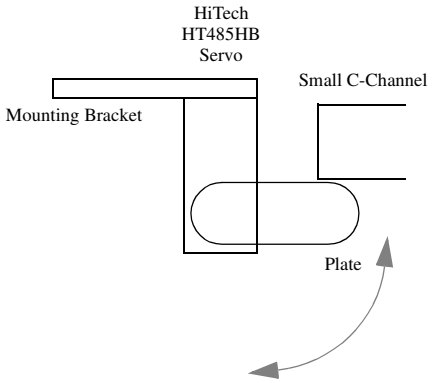

Figure 3 - Foundation Hook Mechanism

Converting Joystick Input To Robot Heading

The Logitech game pad has two joysticks per game pad, which we plan to use for controlling the robot drive system. Each joystick provides a value between -1.0 and +1.0 on both the *x*- and *y*-axies. On the *x*-axis, -1.0 is to the far left, while +1.0 is to the far right. On the *y*-axis, -1.0 is toward the top, away from the player, while +1.0 is toward the bottom, closest to the player. Each position of the joystick gives a point (*x*,*y*) in a Cartesian plane.

The trigonometric function that computes the angle from the *x*-axis to the origin to a point on the plane is the *arctangent* function. In the Java programming language, it is the *atan2* function in the

Math class. Now, because the sign of the *y*-axis is the opposite of what the *arctangent* function expects, we also have to reverse the sign of the *y* input in order to orient it with a standard trigonometric frame of reference. To convert the angle in the trigonometric frame of reference to a bearing relative to the robot, we reverse the sign of the angle and add 90 degrees.

Holonomic Drive System

A holonomic drive system allows the robot to drive forwards and backwards just like a car or a tank. But, unlike a car, it also allows the robot to move left or right without turning. The wheels are oriented in such a way that the force from the rotating wheels pushes in angles that are partly towards and partly away from the direction of travel. The trick is that the forces balance each other *except* in the direction of travel. This may be done by physically angling the wheels, as we have done with Omni wheels, or by designing the wheels to provide the force at an angle, as Mecanum wheels are designed to do.

Omni wheels are turned 45° inwards in the front, and 45° outwards in the back. When the wheels rotate in the normal direction for going forward, that is, counterclockwise on the left and clockwise on the right, the side forces balance each other, pushing the robot forward. Similarly, when the wheels turn in a backwards direction the side forces are balanced, leaving the robot to move backwards. A side motion happens when the two front wheels turn in the same direction and the two back wheels turn in the direction opposite from the front wheels. This is illustrated in the diagrams below.

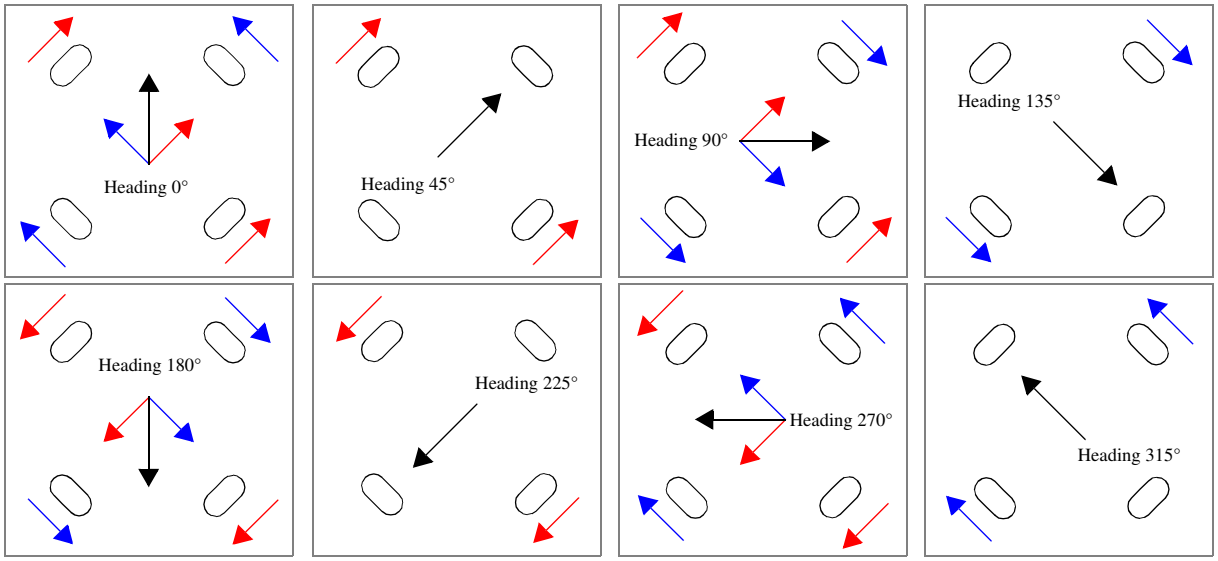

Figure 4 - Holonomic Force Diagrams

In the diagrams, the wheels with the red arrows turn in such a way as to push the robot in the direction of the arrows. The wheels with the blue arrows do the same. Adding the red and blue arrows together gives us the black arrow, which shows the robot's direction of travel.

The next step is to translate the heading we want to travel into power settings for each of the four wheels. We observed that *sine* and *cosine* functions have the behavior we wanted, so we used the

force diagrams (above) with the Pythagorean Theorem to create the following table of headings and corresponding power settings. From that it was easy to identify which function was needed.

Table 1: Wheel Power Settings

Heading α	0°	45°	90°	135°	180°	225°	270°	315°	Function
Front Left	-0.7	-1	-0.7	0	0.7	1	0.7	0	$-sin(\alpha+45°)$
Front Right	0.7	0	-0.7	-1	-0.7	0	0.7	1	$cos(\alpha+45°)$
Back Right	0.7	1	0.7	0	-0.7	-1	-0.7	0	$sin(\alpha+45°)$
Back Left	-0.7	0	0.7	1	0.7	0	-0.7	-1	$-cos(\alpha+45°)$

Using these functions keeps the power setting between -1.0 and +1.0, which is required by the motors.

In the next figure we show the full translation from joystick input to drive motor power settings. The red angles are the heading of travel (or bearing) in radians, while the compass rose in the upper right corner shows the heading in degrees. The angles in black show the angles relative to the joystick. The dark blue square shows the joystick range of motion, while the light blue circle within the square shows the allowable joystick values. The drive motor power formulae are given in the lower right corner of the figure.

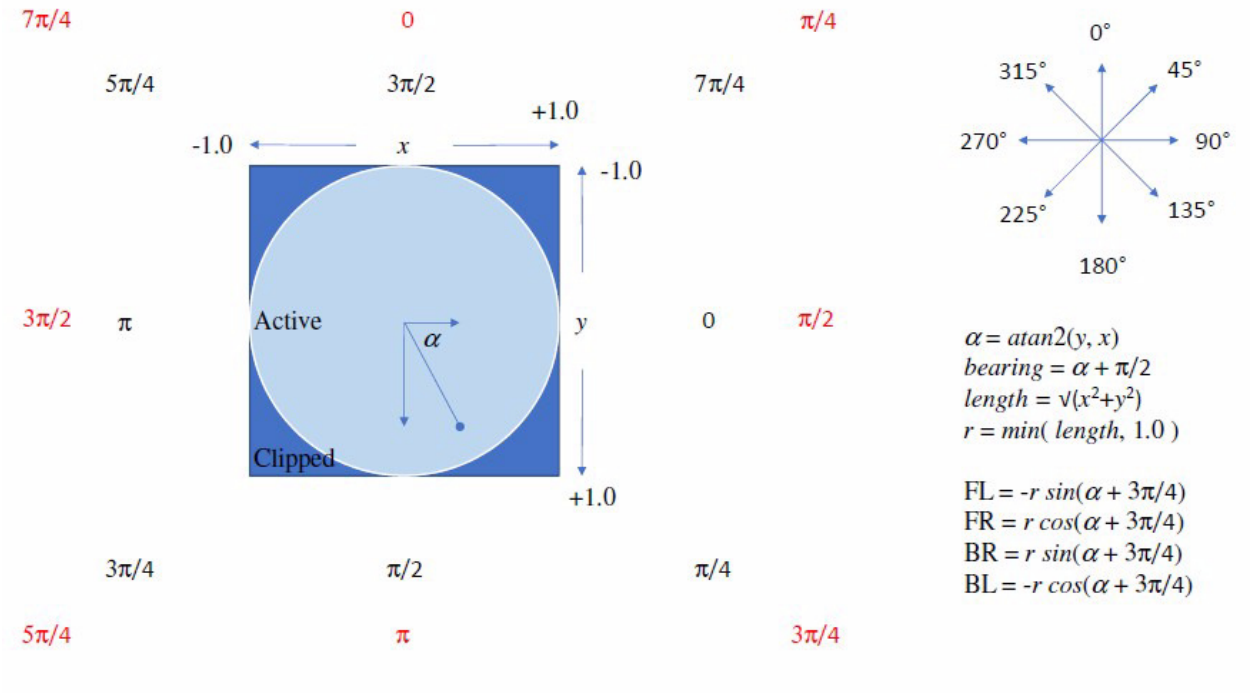

Figure 5 - Translation From Joystick Input To Drive Motor Power

Electrical Design

The robot electrical design is built around two REV Robotics Expansion Hubs. Each hub provides ports for four (9.6v to 12v) motors with encoders, six 5v servo ports, two +5v power ports, four analog ports, eight digital ports, and four I2C ports. There are also two RS485 ports for connecting additional hubs and two UART ports for debugging. (See "REV Robotics Expansion Hub Guide -- Rev 4," http://www.revrobotics.com/content/docs/REV-31-1153-GS.pdf, for details.)

Our robot uses two expansion hubs in its design. The primary expansion hub (Hub 2) is used to control the four drive motors and two (claw and hook) servos. The additional hub (Hub 3) is used to control the lift motor. The ports selected and the names given to each port in the robot configuration file are given in the next figure.

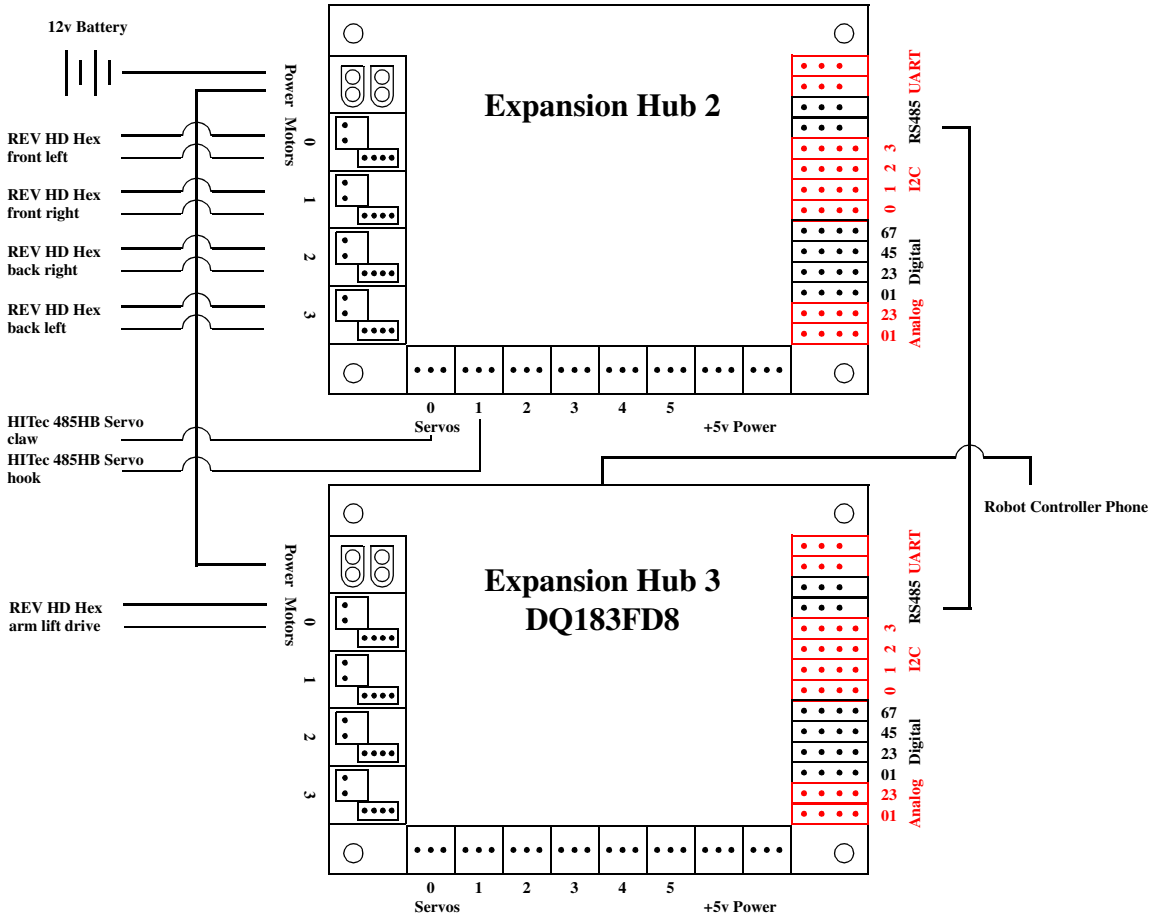

Figure 6 - Electrical Component Block Diagram

Program Design

The program has two main classes of operation, namely, the autonomous operation mode (or "op mode") and the driver (or "teleop" mode). We wrote our op modes in the Java programming

language using the FTC library and Android Studio. There is only one driver op mode, but in order to handle the different requirements of starting in different quadrants on the field, we created separate autonomous op modes for each quadrant (i.e., Blue vs. Red Alliance, and Loading vs. Building Zones).

Program Structure

In previous years our robot op modes have not had any re-usable components. Every year we have started over completely from scratch. This typically required many weeks of programming and resulted in programs that were fragile, error prone and difficult to change. Last year we decided to try a new approach and create high-level program components that could be used from one year to the next with only minor modifications to address the changing requirements of each FTC challenge. We used some of the features of Java to create separate components for the transportation base (holonomic drive system), the arm assembly, sensors, etc.

This year was our first opportunity to determine whether the software we developed last year was truly reusable, and our efforts were spectacularly successful! In past years it took several months to create driver and autonomous op modes. This year it took about one week to port the old code and write all five new op modes. We also found that the reusable program structure made it much easier to try different designs within the same robot.

Our program structure is illustrated in the next diagram. (See Fig. 7.)

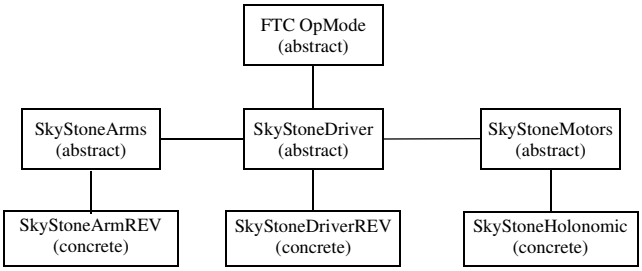

Figure 7 - Program High-Level Structure

Each major function of the robot was used to create an abstraction that reflected *what* the robot would do instead of *how* the robot would do it. For example, our SkyStoneMotors class defined what operations were needed to move the robot around the playing field, in other words *what* the robot does, while our SkyStoneHolonomic class implemented the actual operations themselves, or *how* the robot performs those operations. In this way, changes to the robot were isolated to the concrete sub-classes where that subsystem was implemented. The *use* of those operations or subsystems in the main body of the driver or autonomous op mode were unaffected.

It also allowed the programmers to think in terms of high-level operations for the actual op modes instead of forcing them to constantly focus on details of motor or servo operation. This simplified the op modes to commands such as “move forward 6 inches” instead of complex motor settings and operations. **It was faster to learn, faster to code, and easier to verify it was correct.**

Robot Configuration

The robot configuration is listed in the following table. (See Table 2.)

Table 2:

Hub	Class	Port	Type	Name
1	Motor	0	Rev Robotics HD Hex	front left
1	Motor	1	Rev Robotics HD Hex	front right
1	Motor	2	Rev Robotics HD Hex	back right
1	Motor	3	Rev Robotics HD Hex	back left
1	Servo	0	Servo	hook
1	Servo	1	Servo	grab
2	Motor	0	Rev Robotics HD Hex	lift