Executive Statement

Brought to you by The Charge's programming sub-team

The Charge's control systems are innovative and provide unique capabilities and benefits to our robot. These are some of our most important innovations and their benefits:

Autonomous: Motion Magic	Speed & accuracy in driving a set distance
Autonomous: PID Turning	Speed & accuracy in turning a set angle
Autonomous: Drive to Current	Motor protection & the ability to tell when hit an object
Subsystems: Invert Drive	Makes the back of the robot the front
Subsystems: Joystick Control Modes	Driver comfort for optimal performance
Driver Friendliness: White List	Simple debugging clean up
Driver Friendliness: Button Box	Allows driver to concentrate on driving

The Charge developed and tested our control system in a multi-tier environment. This allowed concurrent development and the efficient execution on rigorous testing. This is what our environment consisted of:

Development Environment	Eclipse
Source Code Management	GitHub & SourceTree
Debugging	Smart Dashboard
Testing	Ply-Bot, Practice Bot & Competition Bot

These central systems have a record of 36 total gears placed, 38 centerlines crossed over a total of 38 district matches. We scored 679 autonomous points at Kettering week one and 870 and Midland. We were the number one alliance captain at week one Kettering and on the number two alliance at Midland. Both of our alliances advanced to the finals. We won week one Kettering. These systems were crucial to our many achievements and for The Charge to end the districts competitions ranked 2nd in Michigan.

Table of Contents

Autonomous

Key Features: Drive Feet, Turning and Current Control

What is Autonomous?

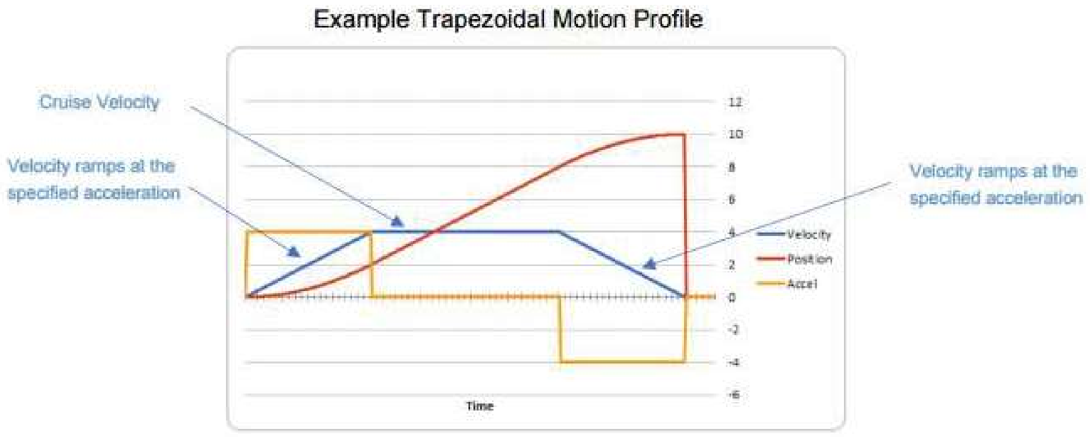
Autonomous is a fifteen second period at the start of every match where the robot is controlled entirely by code, with no driver input.

Drive Feet

This feature of autonomous started as what is called a state engine controller. A state engine controller has a target it is trying to reach. In this instance the target was a distance of feet. However, state engine controlled motors will oscillate around their target until they achieve the desired accuracy. Thus one must sacrifice speed or accuracy with a state engine controller.

The next step in the evolution of our drive feet was a PID controller. PID controllers adjust speed by the error from the target in order to increase accuracy while maintaining speed. However, the PID controller was not as accurate as we would have liked, so the evolution continued.

The final step in the evolution of our drive feet method was a CANTalon feature called motion magic. Motion magic is a new feature on the CANTalons this year that controls velocity and acceleration based on distance to the target. This feature allowed us to be both speedy and accurate with our drive feet method. The final product gets us within inches of the target value. Velocity and acceleration are set in the code along with a target in feet, the overall drive then looks something like this:



Turning

This feature also started as a state engine controller similar to that of our drive feet. However, the target was an angle based off of our gyro. For a full explanation of our gyro see page 14. In essence a gyro keeps track of the total angular rotation of our robot and the position of our robot based on zero. In the gyro we used, zero is defined as the direction the robot was facing when powered on. Again the state engine controller sacrificed speed or accuracy. Thus we progressed to a PID controller for our turning abilities as well.

Our robot can turn in two ways: relative and absolute. Relative turning means that the robot turns 20 degrees to the right of where it currently is. Absolute turning means the robot turns to 20 degrees relative to the gyros zero.

In the code absolute turning reads a method from the gyro library that returns the position of the gyro relative to zero in order to see if it has reached its target. The relative turning however, sets the cumulative angular rotation to zero and then reads the cumulative rotation to see if it has reached the target angle.

For autonomous we used absolute turning because there is no cumulative error, ensuring greater consistency and accuracy. Since absolute turning is based on the gyro's zero it is pertinent that the zero is in the correct location for autonomous. In order to save our drive team grief, we included a function that sets the zero of the gyro to the direction it is facing at the start of autonomous.

Current Control

The ability to limit and measure the current drawn by our motors has been preeminent in the success of our robot. Current limiting is important for the efficiency and lifespan of our motors during a robotics competition. As a motor applies force against an obstacle, after a certain point, the motor will stall, additional current will not increase the speed and torque of the motor, and any additional current will be converted into heat, which can significantly shorten the lifespan of the motor. By simply limiting the current of our motors through the Talon SRX we are able to prevent motor failures and save huge amounts of battery voltage.

Measuring the current drawn by our motors is important for controlling our robot during autonomous. By monitoring the current of a motor we are able to detect when our robot us pushing up against the airship, or the boiler. This feature increases the accuracy of our autonomous, by accounting for any error that may have occurred in previous parts of the autonomous code.

Subsystems

Key Features: Drive Train, Climber, Indexer, Shooter, Collector, Pneumatics

What is a Subsystem?

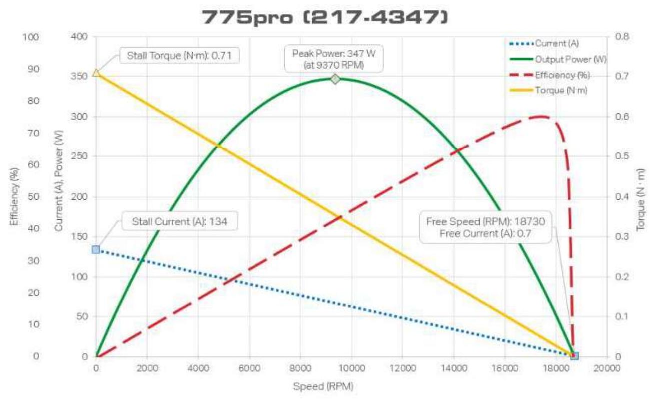
Subsystems are a feature used by The Charge to group different mechanical aspects by function in the code.

Climber

This subsystem is all things related to the climbing mechanism on our robot. A unique feature of our climbing subsystem was its digital input. A digital input returns true or false to the code based on its physical state of pressed or unpressed. The digital input hits the press pad just before our climbing mechanism, and sends a value to the code. This stops the climber's motors and allows us to prevent severe damage of game pieces.

Another unique feature was the style of our override button. In order to save our drivers time if the digital input got triggered early and to prevent breaking the game piece, this button functioned by both a timer and pressing of the button. It was done by checking the value of the button pressed and a 0.05 second timer. If either of those things were true the climb motor could continue to run.

The third unique feature here was current limiting. There were many times where we would have fried the motors without this features. The graph below is the motor curve of the 775 Pro, our climbing motor. Current limiting effectively cut the graph off just past the peak of the green curve. This featured prevented any possible stalling or breakage of the motor due to obscenely high amperage.

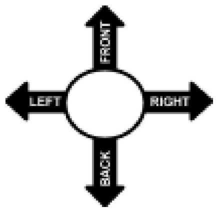
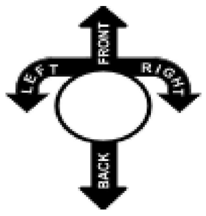
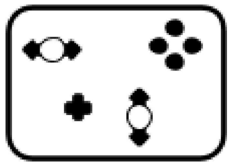
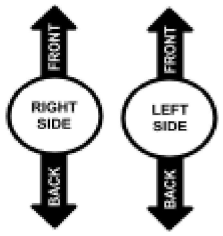
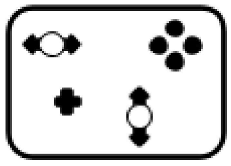


Indexer

This subsystem was related to the indexer on our robot. An indexer feeds balls into the shooter and stops balls from jamming in the shooter. The unique feature here was another digital input. This input was attached to the robot just outside of the indexer's motor. It was used to stop the indexer in a favorable location. Effectively after we physically told the indexer to stop, it would keep running until the digital input returned true to the code. The extra run time was only a second at most, but it made sure balls wouldn't roll into the shooter while it was off and hamper shooting the next time we shot.

Drive Train

This subsystem was all things related to the wheels of our robot. A unique feature of our drive train were the many different joystick control modes.

	Controller Type	Number of Controllers	Diagram
Arcade Drive	Joystick	1	
Clayton Drive	Joystick	1	
Halo Drive	Game Controllers	1	
Tank Drive	Joystick	2	
XBox Drive	Game Controllers	1	

Another unique feature of the drive train subsystem is the invert drive feature. This feature was created for the safety of our robot, time conservation and driver ease. Invert drive keeps the controls for going forward or backwards the same, but the back of the robot is now the front. This keeps the frame from accidentally being bent during pushing because we can push with the back of our robot and have the same maneuverability as with the front. This feature conserves time because our driver doesn't have to turn around to go across the field or get out of being pushed. It also makes driving backwards much easier for the driver because they don't have to think like they are going backwards.

A third unique feature of the drive train is combined low gear and speed control mode. For definitions of speed control and low gear respectively see the shooter and pneumatic subsystems. This speed control is unique. In the code it is bounded with a max of quarter speed. Combined these two features allow for an accurate, smooth control in delicate locations such as delivering a gear.

Shooter

This subsystem had all things to do with our shooter. The unique feature here was speed control mode. This is a mode on the CANTalons that uses target speed set in the code. The CANTalon will then regulate the motor so it runs at that speed consistently. The advantages to using speed control mode are that the motor will slow down less if a ball gets stuck in the shooter and recovery to the set speed is faster if a ball does get stuck. This control mode helped to ensure our shooter ran consistently.

Another unique feature was the manual override for this set speed. In case the shooter was not getting balls in the boiler we included a dial that allowed the drivers to control the shooter speed if necessary. This dial has two unique parts to it. First the dial was not taken into effect unless the drive team flipped another switch to allow manual override of the set speed. Secondly the dial only allows the drive team to tune the speed within +/- 10 of the set speed. So if the set speed was 80 the drivers could tune it to be between 70 and 90 with the manual override. These features were enacted to give our drivers fine control over the speed of the shooter when necessary.

Collector

This subsystem controlled anything to do with our robots collecting mechanism. The unique feature here was speed control. Speed control for the collector created the same benefits in the collecting mechanism as it did in the shooting mechanism.

Pneumatics

This subsystem controlled all of our pneumatic devices on the robot. There were three in total: doors, plunger and shifters. The pneumatically operated doors allowed us to keep a gear in our gear holding mechanism until we had said gear on a peg. The plunger allowed us to push the gear to the back of the peg so pilots could bring the gear up into the airship easier. Thanks to the shifters our robot is in either high gear or low gear. In essence low gear gives us more accuracy and high gear gives us more speed. Each of these gears is used in a different part of the game. Low gear ensures accuracy in autonomous and putting gears on the peg. Low gear also works great for pushing other robots across the field. High gear allows us to drive quickly across the field.

CANTalons

Key Features: CANTalon Modes, Efficiency and Web Browser Interface

What is a CANTalon?

CANTalons are a type of motor controller. Motor controllers allow programming teams to control different features of running a motor, such as speed. The specific type of CANTalon used by The Charge is a Talon SRX. CANTalons are produced by Cross The Road Electronics.

Follower Mode

Follower mode is a feature of the Talon SRX that simplifies the code. When in Follower Mode, a motor controller will duplicate the output of another motor control. We used Follower mode for features such as the right and left sides of our robot's drive train. Each side of our drive train was controlled by two motors. Thus we only had to set a target speed for one of the right motors and the other right motor would match the first.

Invert Mode

Invert mode is a feature of the Talon SRX that negates the need to remember which motors drive in reverse or forward when coding. For example if a motor in invert mode is sent a 1 the motor translates that value to a -1 automatically. This feature comes in handy when motors are wired in reverse. Invert mode allows the programming team to work with this issue much more efficiently than going into the code and hardcoding a negative sign in front of every setpoint given to that motor.

Brake Mode

Brake mode is a feature of the Talon SRX that is extremely important for the accuracy of our autonomous. Brake mode is also used on almost every motor in our robot. When brake mode is enabled it works in the background of the Talons to prevent coasting as soon as the code tells the motor to stop. This greatly improves the accuracy of all features of the robot.

Motion Magic Mode

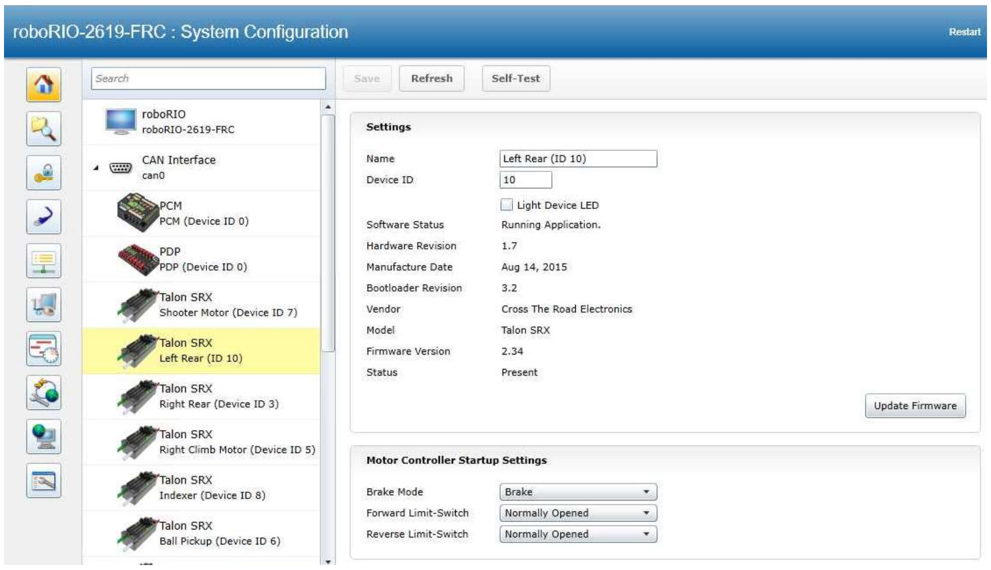
This feature was used primarily in the autonomous code. See page 2 for an explanation.

Percent VBus

Percent VBus is a CANTalon motor mode that operates as our default mode for driving. Percent VBus sends a percentage of voltage to the motors. We determine that percentage by joystick input. Percent VBus is our choice of default driving mode because of the simplicity and functionality it provides. Functionality wise percent VBus provides full forward to full reverse and everything in between. While this is similar to other control modes on CANTalons, percent VBus doesn't require any sort of PID values. Lack of PID values simplifies the code, making percent VBus preferable.

Web Browser Interface

The web browser interface is a feature of the RoboRIO, the center of communications for the robot. It allows us to see what is happening with the motors as the robot moves. The web browser also allows us to see items like the current mode of a motor. On the right is a picture of the general web browser and the left is a the specifications for one motor.

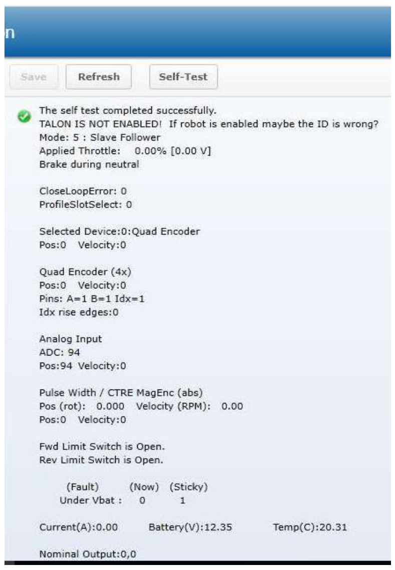


Settings

Name	Left Rear (ID 10)
Device ID	10
	Light Device LED
Software Status	Running Application.
Hardware Revision	1.7
Manufacture Date	Aug 14, 2015
Bootloader Revision	3.2
Vendor	Cross The Road Electronics
Model	Talon SRX
Firmware Version	2.34
Status	Present

Motor Controller Startup Settings

Brake Mode	Brake
Forward Limit-Switch	Normally Opened
Reverse Limit-Switch	Normally Opened



```
The self test completed successfully.
TALON IS NOT ENABLED!  If robot is enabled maybe the ID is wrong?
Mode: 5 : Slave Follower
Applied Throttle:   0.00% [0.00 V]
Brake during neutral

CloseLoopError: 0
ProfileSlotSelect: 0

Selected Device:0:Quad Encoder
Pos:0   Velocity:0

Quad Encoder (4x)
Pos:0   Velocity:0
Pins: A=1 B=1 Idx=1
Idx rise edges:0

Analog Input
ADC: 94
Pos:94  Velocity:0

Pulse Width / CTRE MagEnc (abs)
Pos (rot):   0.000   Velocity (RPM):   0.00
Pos:0   Velocity:0

Fwd Limit Switch is Open.
Rev Limit Switch is Open.

      (Fault)       (Now)   (Sticky)
     Under Vbat :     0         1

Current(A):0.00       Battery(V):12.35        Temp(C):20.31

Nominal Output:0,0
```

Efficiency

CANTalons bring the advantage of a smaller size. Talons are 2.75" wide and 1.2" long. This means they take up roughly 3.3 square inches of space on the robot. When compared to some of The Charges past motors such as Jaguars they are very space efficient. Jaguars are 4.25" wide, 3.8" long and take up a grand total of 16.2 square inches. The space efficiency of Talons means that less space on the robot has to be delegated to electronics, allowing more features.

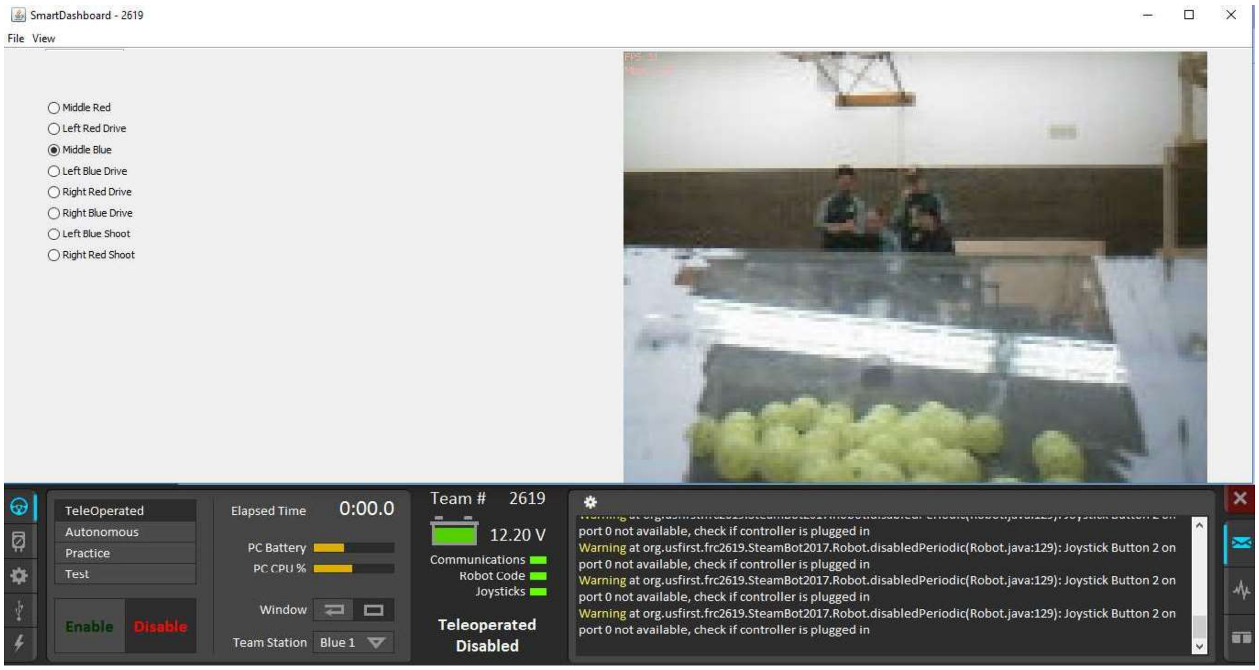
CANTalons also have significantly faster processing speed than other motor controllers The Charge has used throughout the years, including Jaguars. This increases efficiency too.

Driver Friendliness

Key Features: Smart Dashboard and Button Box

Smart Dashboard

The smart dashboard is an user interface with the code and the robot. This is the platform we use to debug and test code as well as select autonomous' and view cameras during the game. This is a screenshot of our competition smart dashboard:



○ Middle Red
○ Left Red Drive
● Middle Blue
○ Left Blue Drive
○ Right Red Drive
○ Right Blue Drive
○ Left Blue Shoot
○ Right Red Shoot

```
port 0 not available, check if controller is plugged in
Warning at org.usfirst.frc2619.SteamBot2017.Robot.disabledPeriodic(Robot.java:129): Joystick Button 2 on
port 0 not available, check if controller is plugged in
Warning at org.usfirst.frc2619.SteamBot2017.Robot.disabledPeriodic(Robot.java:129): Joystick Button 2 on
port 0 not available, check if controller is plugged in
Warning at org.usfirst.frc2619.SteamBot2017.Robot.disabledPeriodic(Robot.java:129): Joystick Button 2 on
port 0 not available, check if controller is plugged in
```

A unique feature of the smart dashboard is the sendable chooser, the list of buttons on the left of the picture. The sendable chooser allows our drive team to select any of our eight autonomous option. The sendable chooser provides many benefits. First of all it allows us to have multiple autonomous options, which creates flexibility for our robot based on alliance partners and strategy. Secondly sendable choosers are easily labeled so drivers can tell what they are selecting. Past years we have used buttons on the button box or joysticks, but sendable choosers negate the need for the drive team to carry around any sort of cheat sheet to find the correct pattern of switches.

Another unique feature of the smart dashboard is the white list. This was created by The Charge last year. A white list is a collection of all the strings used to put buttons, inputs and outputs that have been written to the smart dashboard for debugging. The string for individual objects must be included in the white list in order for it to be displayed. The white list allows us to write debugging material to the dashboard and easily remove it later. Without the white list, there are over 57 buttons and 70 outputs on the screen. Suffice to say, they don't all fit.