Stormgears 2016 Advanced Control System

White Paper

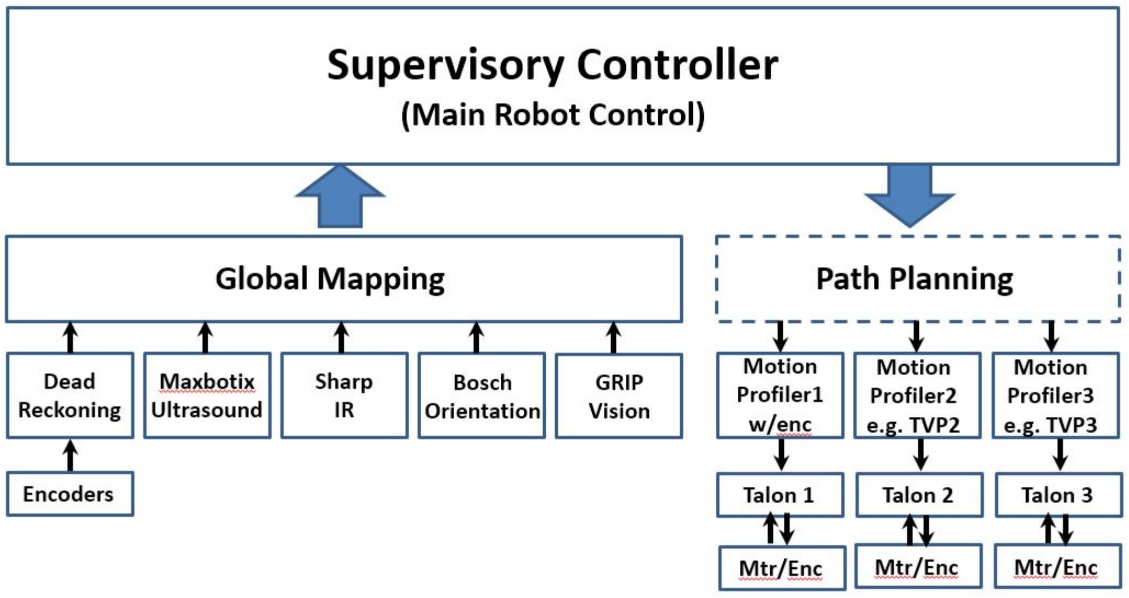
By Team 5422

Electrical, Controls, and Software Sub-Teams

Introduction

For the Stronghold 2016 FRC Competition, Stormgears has employed an advanced control system architecture that we will continue to build on in future seasons. At the heart of our approach is a hierarchical control strategy in which we maintain a software model of the game field, and keep track of our robot position in real time in field coordinates (x, y, and theta). Every motor on our robot has closed loop PID control using Talon SRX motor controllers and encoder (incremental or absolute) feedback. Then we generate motion profiles in the RoboRio based on desired commands from autonomous routines, algorithm-assisted teleop routines, and teleop input controls (joystick, sliders, knobs, buttons).

The global position of the robot in field coordinates is maintained via odometric feedback from the drive motors (CUI digital quadrature encoder feedback). Over time, the positional accuracy drifts due to slippage and left/right drive system variances (wear, friction, wheel diameter). Context specific sensor readings from a diverse suite of sensors are utilized to correct the error and re-register the global position. The sensory suite includes encoders for drive motors, potentiometer for lifter motor, gyro/accelerometer for theta correction after turns and after traversing defenses, downward-looking infrared sensors for tape detection and y-theta correction, forward-looking ultrasounds for y-theta correction, and vision for theta alignment and radial distance measurement to support boulder shooter aiming. The sensors and applicable Stormgears software control strategies are described herein.

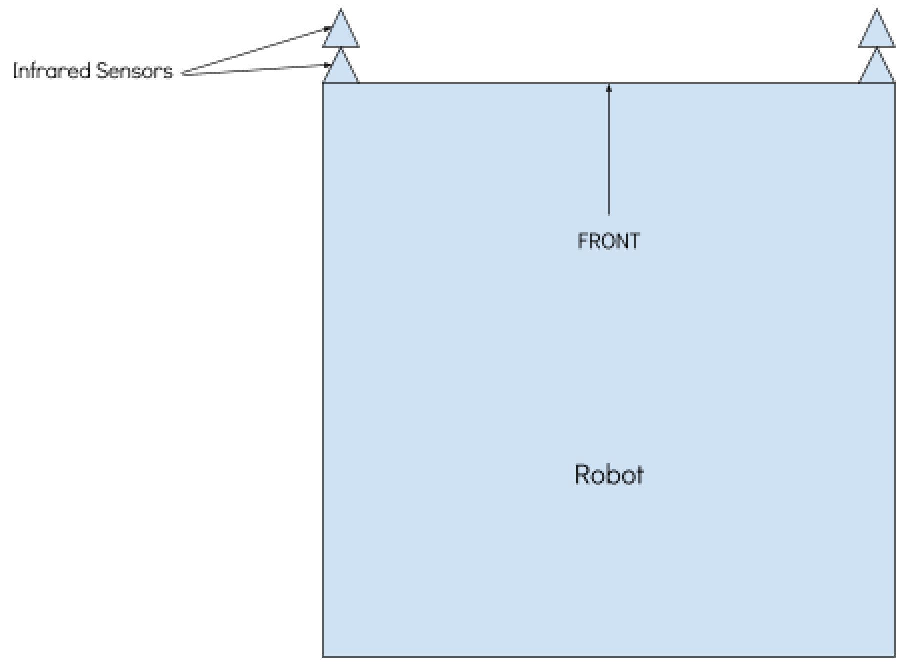


MXP Circuit

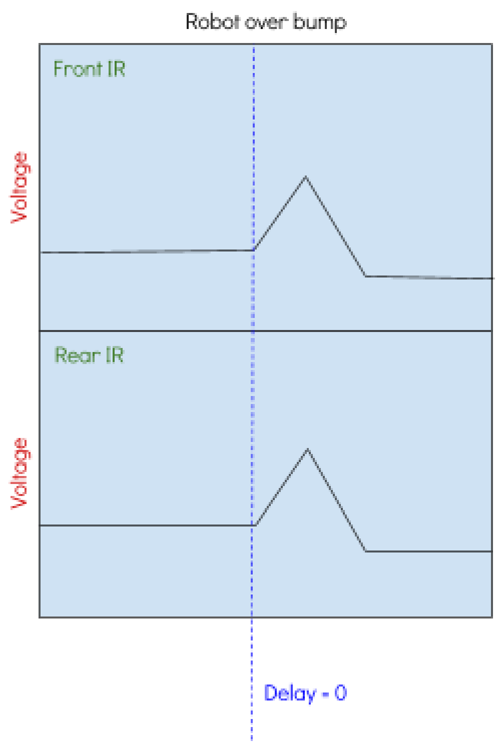
We have built a single custom circuit to house all interface circuits and sensor inputs to the RoboRio. This circuit is located on the RoboRio plugged into the MXP port. Details on its components and their uses are as follows in the proceeding sections.

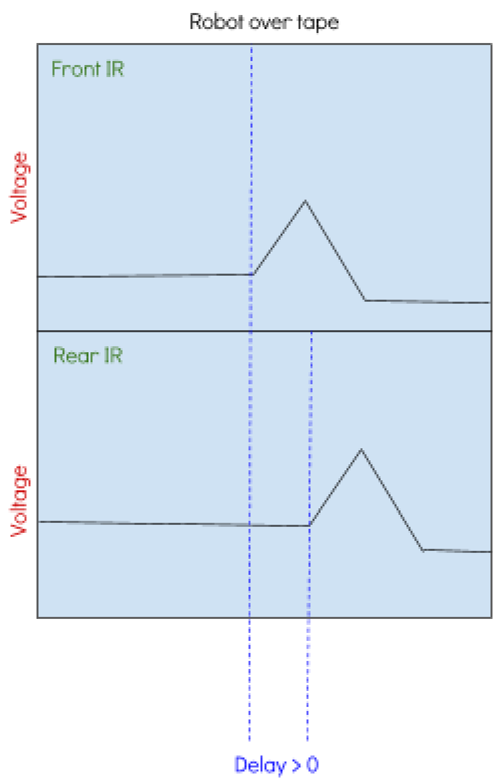
Infrared

Use: We are using four Sharp GP2Y0A21YK0F infrared sensors mounted in two sets of two at each front corner of the robot. The sensors are facing down about 2.5 inches above the floor. The pairs are mounted together, one in front of the other.



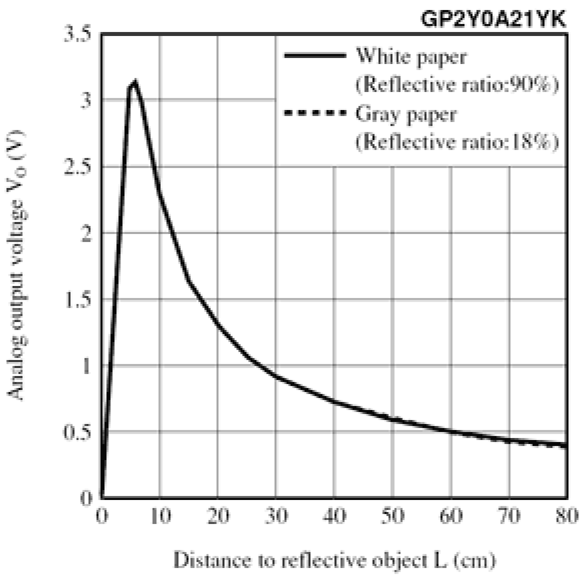
We are using the inputs to detect the green marking tape on the field. Knowledge of our robot's relation to the tape can provide an extremely accurate external sensory input to our global positioning system. By having data from both sides of the robot, we can calculate our orientation on the field by comparing the drive encoder values at the time of the detection of the tape on each side.





In preliminary testing, we determined that the voltage difference between the carpet and the tape is low enough where it could be difficult to detect reliably with a single infrared sensor. To solve this, we have paired the sensors and only pay attention to the difference in their voltages. If the robot were to bounce or tilt, both sensors will read the same change in voltage simultaneously. When we cross the tape on the field, however, one sensor will change before the other one does, thus detecting the tape.

Data Input: Each sensor outputs a voltage ranging from 2.3V which corresponds to 10 cm to 0.4V which corresponds to 80 cm. The relationship between voltage and centimeters is nonlinear, and graphed below.



In preliminary testing, we also tested two shorter range IR sensors to determine the best sensor for detecting the tape with the largest change in voltage. We determined that the long range sensors were able to detect the tape the best due to the greater difference in voltage among smaller distances.

Connections: We used the built-in connector on each sensor to connect to ports IR0, IR1, IR2, and IR3 on the custom MXP circuit. A locking .100" connector was used to attach the sensors to labelled headers on the MXP circuit. On the circuit, power and ground is provided to pins 2 and 1 respectively on each IR input. Pin 3 is passively connected to either AI0, AI1, AI2, or AI3 for the respective ports.

Light Control

Use: In order to implement Graphical Image Processing (GRIP) we used a set of three LED ring lights. These ring lights are mounted to our camera by a custom designed 3D-printed mount. Our software team can turn these lights on specifically when we are shooting and turn them off afterwards for battery efficiency and to avoid driver interference. The lights on our camera allow us to use GRIP in order to align ourselves with the high goals that are presented in the 2016 FIRST Stronghold challenge. We have also implemented decorative tractor lights on our drive train which can also be controlled by software.

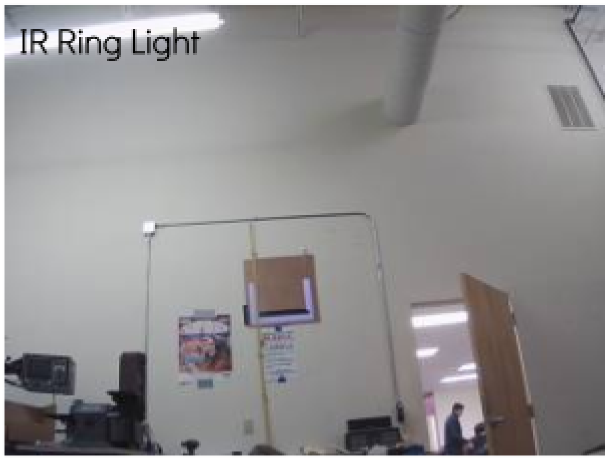
Connections: The three nested ring lights are wired in parallel and connected to either port L1 or L2 on the MXP circuit. Because the ring lights and the tractor lights require a 12V output, but the MXP circuit only supplies a 3.3V output from the DIO pins, we created a custom light control circuit. We also supplied the custom MXP circuit with a 12 volt power supply from a 12 volt VB3

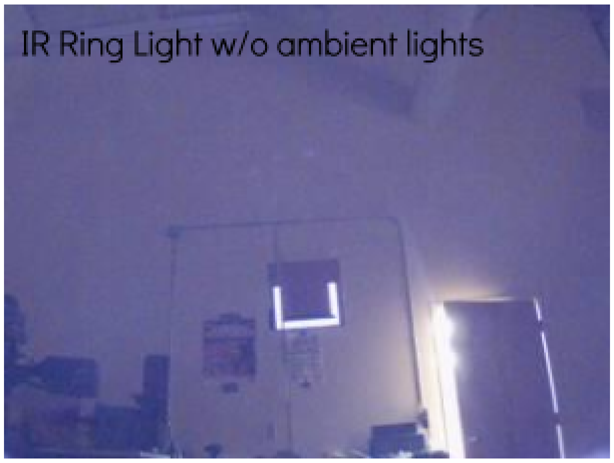
Breaker output on the power distribution panel. This supply is connected to port P1 on the MXP circuit.

Custom Circuit: Our automatic aiming vision system relied on illuminating the retroreflective tape from maximum distance, meaning we had to use three nested LED ring lights at high brightness. Combined with our decorative tractor lights, this posed a problem for conserving battery power during the match. To solve this problem, we designed and built our own custom dual channel light control circuit, which would allow us to switch the high currents these lights draw (1.5 amps continuous) from a low current roboRIO digital output, conserving battery power by only illuminating the tape when we needed to take vision measurements and giving us the option to turn the decorative lights on and off.

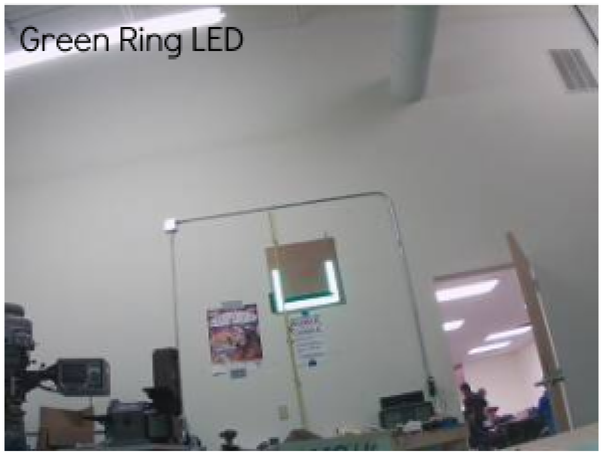
To enable the software team to control the lights remotely, we used 3.3V MOSFETs which we could control with the digital I/Os on the roboRIO. We powered the lights with 12V from the PDP, which would have to go through the MOSFETs. This way, the lights would only get 12V if software raised the correct DIO pins on the roboRIO to 3.3V.

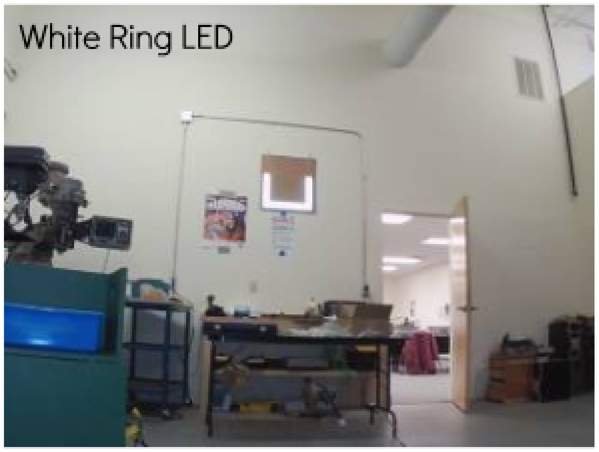
Tests: We ran tests with green 60mm, 80mm, and 100mm nested LED Halo Headlight Accent ring lights, and custom built infrared ring lights. We tested both UT 1883 and brighter SFH 4550 infrared diodes, wired in parallel. Because our AXIS M1013 Network camera is capable of detecting infrared light waves, we theorized that infrared lights may be easier to filter out in our GRIP image processing. After testing, we concluded that both sets of lights are too dim to be used practically on the field.





When testing the green lights, we found that they performed extremely similarly to the white lights. Thus, either color can be filtered similarly by our GRIP image processing. Before the test, we figured that green lights may be easy to filter out within GRIP. However, the AXIS M1013 Network camera's limited color saturation and the fact that all white light also contains green light waves made it difficult to filter out only green light waves. The green color has been concluded to have no practical advantages over white lights in our configuration.





Encoder

Use: On our robot we decided to use CUI AMT102 (for the shooter) and CUI AMT103 (for the drive) Modular Incremental Encoders. We used these encoders in order to have smoother control over all of our subsystems. On our custom tank drive we used the encoders in order make our global positioning system. With the encoders we are able to track our robot anywhere it is on the field. Along with that we are using the encoders to keep track of the RPM at which our shooter wheels are spinning to make sure we get tight control over the speed in order for the ball to shoot properly. These encoders are connected to our robot using the Talon SRX Motor Controller. This allows us to tune our PID values more easily with the use of the National Instruments Utility.

Data Input: Both the CUI AMT102 and the CUI AMT103 encoders input quadrature digital signals to the Talon SRX Universal Breakout Boards. The Talon SRX then keeps a 12-bit counter of the ticks as the encoders rotate. Our software team is then able to read this number without dealing with the raw square waves of the encoders' output. They can then read and tightly control the encoders directly on the Talon SRX interface.

Connections: We use black Molex 50-57-9405 connectors for the CUI AMT102 encoders and red AMP 3-640440-5 connectors for the CUI AMT103 encoders. These then connect directly to the 5-pin encoder port on the Universal Breakout Board using the red AMP 3-640440-5 connectors.

Potentiometer

Use: We use a 10K single-turn linear precision potentiometer to control the level of our articulating shooting and intake arm. Mounted on the pivot of the arm, the potentiometer can measure the rotation and position of the arm. Maintaining tight control over the movement and position of this arm allows us to consistently angle the arm for the perfect shot.

Data Input: The potentiometer outputs a voltage ranging from 0 to 3.3V (linearly) for a full 340 degree rotation. The breakout boards we are using actually predispose the input to 5 volts. This actually changed the amount of ticks we received from the potentiometer. In order to maximize the range of the potentiometer we had to change the jumper that was on the breakout board to 3.3 volts. This allowed us to get the full 340 degrees without any free turning space in the pot. We have calibrated the values to the lower and upper limits of our arm's range of motion with the exact voltages. We can then deductively calculate the angle of the arm. The values of the potentiometer are monitored by the Talon SRX for tight PID control within the motor controller. The RoboRio is also reading the values in order to send position commands to the Talon SRX.

Connections: We use a single PWM connector that goes to the 3-pin analog port on the Universal Breakout Board to read the potentiometer values.

Ultrasound

Use: We have two forwards facing MB1040 LV-MaxSonar EZ4 ultrasound sensors and two side facing MB1040 ultrasound sensors. We are using the front facing sensors to find our distance from a wall (usually the back castle wall) as well as our rotation in relation to the wall. This can then be used to recalibrate our global positioning system. The side facing sensors are used to find our distance from a different static wall to recalibrate global positioning.

Data Input: The ultrasound sensors output serial data in RS-232 format, but with 0-5 volts. For each distance reading, occurring every 20 milliseconds, 8 bits of data are sent with the last bit a stop bit. The data is read in 5 ASCII characters, beginning with a capital R, followed by 3 digits which tell the distance, and ending with a carriage return. The three digit number represents the distance found by the sensor in inches. This distance has integer precision and has a range from 6 inches to 255 inches.

Connections: Each sensor is connected using locking .100" connectors to the respective US0, US1, US2, and US3 headers on the MXP circuit. Each ultrasound output port is connected to our multiplexor circuit which is ultimately fed into the UART connector on the MXP port. Within our circuit, because the serial data sent in 0-5 volts and the UART needs to see the data in -12 to 12 volts, an inverter was added to change the voltages to roboRIO readable data. Then because there is only a limited amount of UART ports on the roboRIO MXP board, a 4-1 multiplexer was added to reduce the amount of data inputs to only 1 connection. Because only 1 ultrasound would be used at one time, to reduce cross firing, the multiplexer efficiently solved the problem. To control whether to fire the ultrasound, the roboRIO sends outputs from the MXP boards. To only use one output port a 1-4 multiplexer was added to control all 4 ultrasound

Motor Controllers

Use: For motor controllers on our robot we used the Talon SRX. These motor controllers are CAN compatible, which allows each of our motors to communicate with each other. In return we can control each of our motors alone or set up a Master/Slave protocol, which means that one motor can follow another one. The Talon’s also allow for closed loop communication. This means that we can control every aspect if the robot’s motion. This also helps noise prevention because the motor controllers are also more spread out.

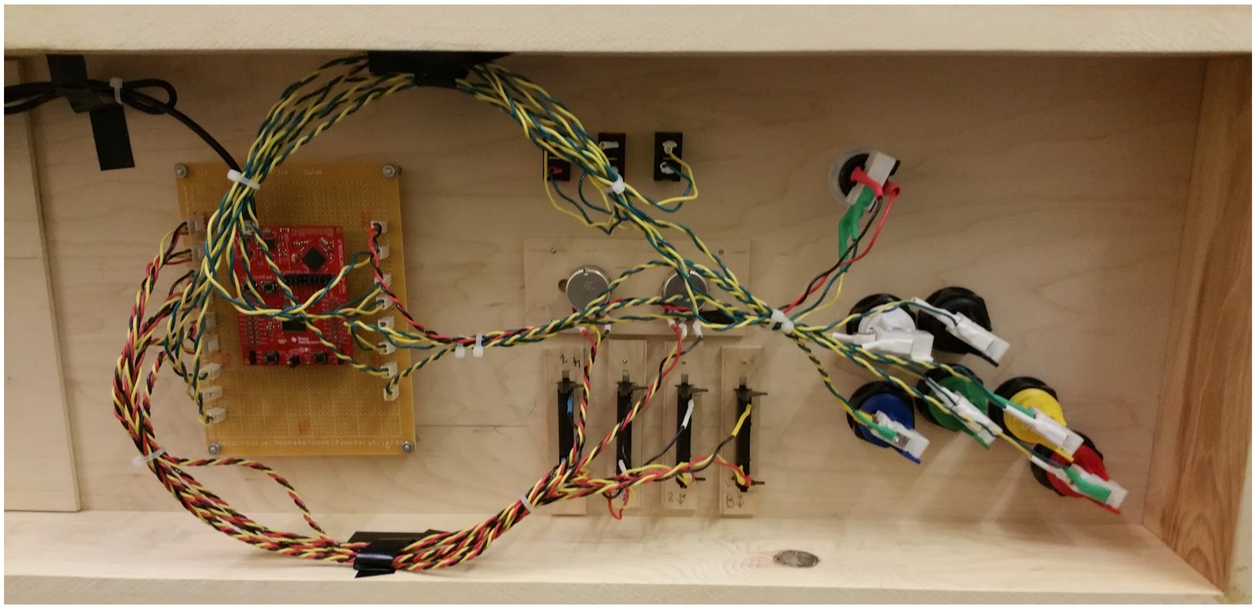
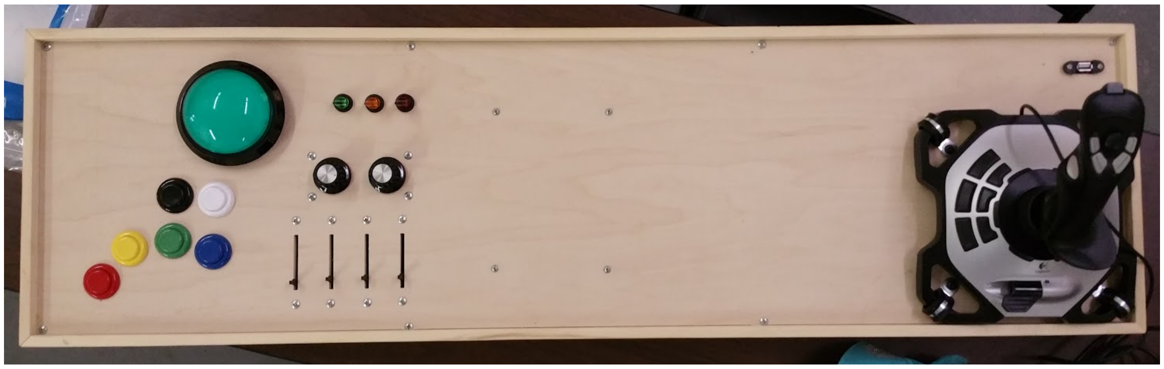
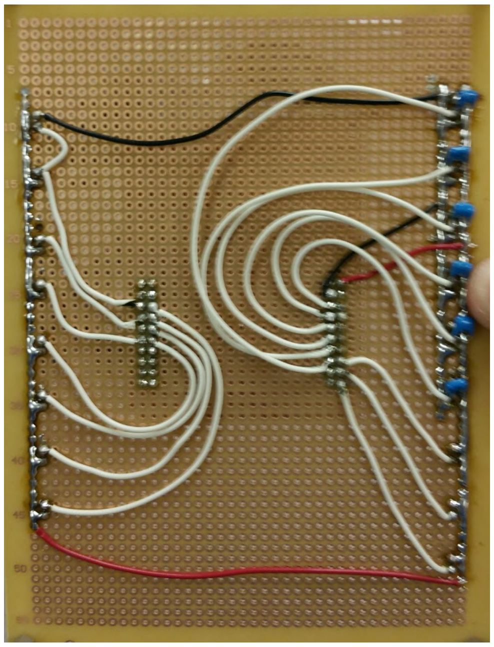
Data Input: The Talon SRX’s can receive data directy from the encoders.

Connections: The Talons’ are connected to the RoboRio and Power Distribution Panel via 22 gauge wire.

Button Board

Use: Our custom button board provides us with a compact and versatile panel of controls when driving the robot. We have designed the ergonomic layout, and software implementation from scratch. Using this button board, we are able to assign specific robot controls to each button, slider, knob, or switch, so we can offload some of the robot functions from the main driver. Our button board allows us to create a secondary driving role to control all aspects of the robot (apart from driving) intuitively and efficiently. We have allocated buttons to autonomous aiming, shooting, and crossing defenses. We have also assigned sliders to manual shooter speed control, and shooter angle. The switches allow us to switch between manual and software-assisted shooting, and turn the intake mode on and off.

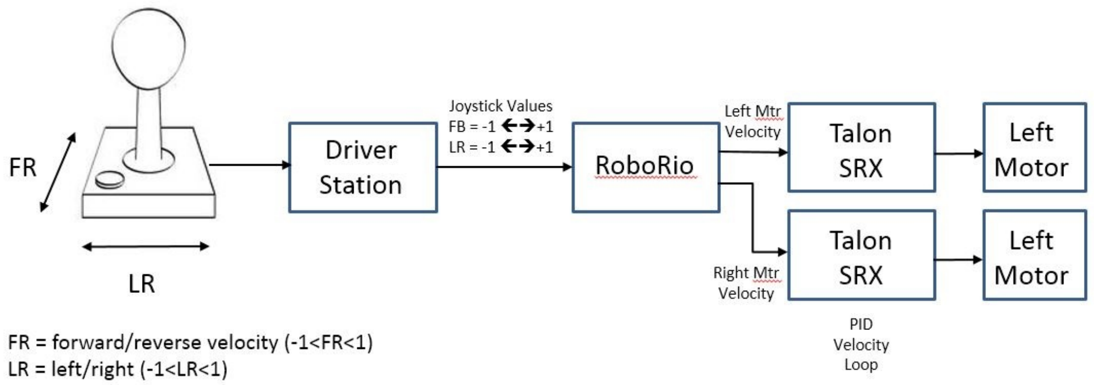
Design: The board is 41 by 11.5 inches with all buttons, sliders, and knobs on the left side of the board. This keeps the secondary driver’s controls all in one place. There is a space for the driver laptop between the buttons and the Logitech Extreme 3D Pro joystick located on the right side of the board. This allows both drivers to see information or camera feeds easily. We have also designed the layout of the buttons ergonomically, so the secondary driver does not need to ever look down during the match to find a button. This allows our drivers to fully focus on the game and intuitively control the robot. We have a total of 7 digital push buttons, 3 switches, 4 sliders, and 2 knobs. The robust design and build of the board will allow us to use this for years to come.

Connections: The board is controlled by a MSP430F5529 Launchpad which connects to the driver computer via micro USB to USB. This board is is set to the option 2 configuration which provides us with enough analog input pins for sliders and knobs, and enough digital input pins for buttons and switches. Each button, switch, knob, and slider is hard-wired to a locking .100" connector which goes to the board-mount header on our custom circuit board connecting to the Launchpad.

Custom Arcade Drive

Our custom arcade drive uses two axes of a joystick to accomplish the forward, reverse, left, and right movements of the robot. The forward and reverse movements of the robot correspond to the Y axis of the joystick. Positive values are forward, and negative values are reverse. The left and right movements of the robot correspond to the X axis of the joystick. Positive values are right, and negative values are left. In order to turn the X and Y coordinates of a joystick into velocity commands for the left and right motors on the robot, we use an arcade drive algorithm. When the joystick is positioned in the top right corner, both X and Y are +1. To translate this into a velocity, both sides of the robot's drive train would begin with a velocity of 1. Then, since the joystick is full right, the absolute value of the joystick's X position is added to the left side. The opposite is true for turning left, as the right side would need to be sped up. For reverse movements, the same thing is done, except all Y values are negative, and the negative absolute value is added to the needed side of the motors. These values are then applied with filters. We have a nullzone turning the first 10% of joystick range into zero, which prevents accidental movement of the joystick. The output of the filters, left velocity and right velocity, is then multiplied by some constants that convert the numbers into velocity commands for the motors, which are run in a closed PID control loop.



FR = forward/reverse velocity ($-1<FR<1$)
LR = left/right ($-1<LR<1$)

K_L = left turning constant ($0<K_L<1$); slows down left wheel when turning left
K_R = right turning constant ($0<K_R<1$); slows down right wheel when turning right
Lv = left motor velocity (-MaxVelocity<Lv<MaxVelocity)
Rv = right motor velocity (-MaxVelocity<Rv<MaxVelocity)

if LR<=0, K_L=1+LR and K_R=1
If LR>=0, K_L=1 and K_R=1-LR
Lv= FR * MaxVelocity * K_L
Rv= FR * MaxVelocity * K_R

***Eliminate "peeling out"**: impose maximum acceleration by capping ABS ($Lv_n - Lv_{n+1}$)*

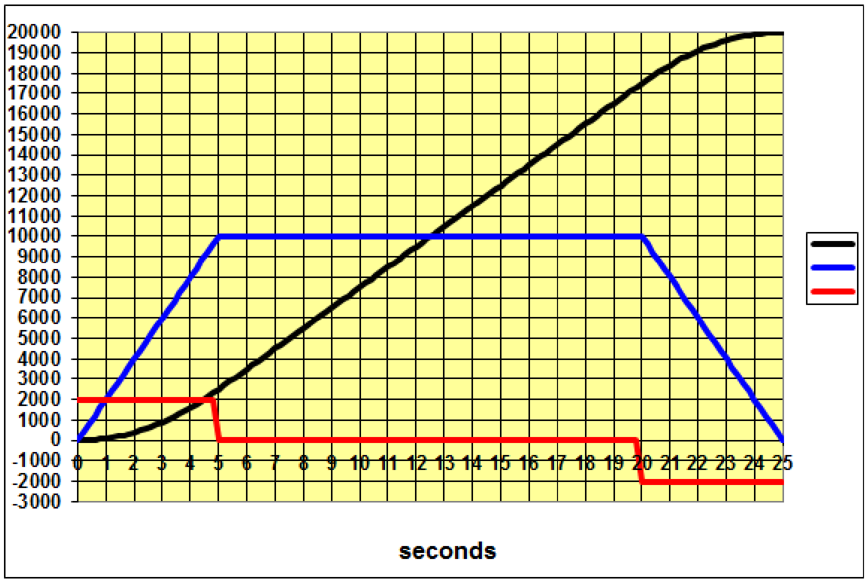
***Enhance fine motion adjustments**: impose a lower maximum acceleration at low velocities and/or apply a damping factor (in range 0%-100%) to delta velocities.*

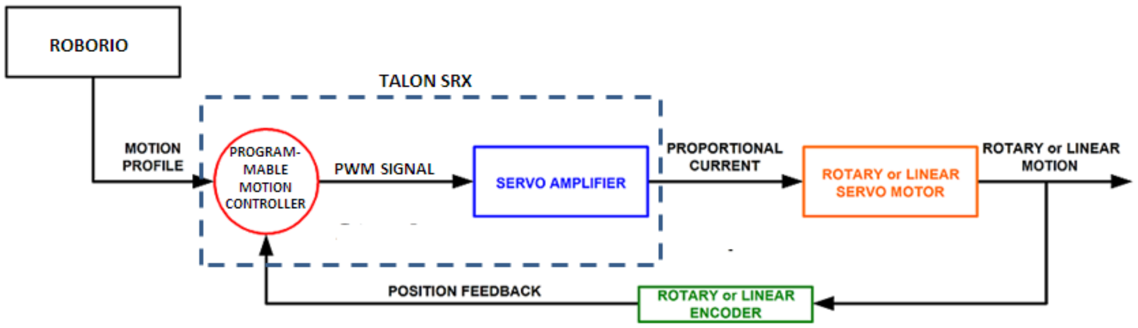
Trapezoidal Motion Control

The traditional way of running a robot controller in closed loop is PID control. Through PID control, a specific setpoint is set on whichever sensor is being utilized, and a PID controller is tuned so that the mechanism can physically get to the setpoint as quickly and accurately. However, in many cases, with PID control it is difficult to achieve the optimal tuning necessary for precise performance. Even the very best tuning could result in small errors in final position of the mechanism, or an extraordinarily long amount of time taken to achieve the given position. In the context of our robot drive, in the short 15 second autonomous mode, speed is crucial, and even slight errors in the position of the robot can lead to failure.

An alternative to traditional PID control is motion profiling. A motion profile is basically a very specific trajectory of numerous setpoints that is provided to the controller. Instead of just giving it one setpoint and telling it to go there, with a motion profile, at any given instance, the supposed position of the controller is known. This basically allows for much tighter control. Specifically, for our robot drive, running trapezoidal velocity profiles allows us to specify parameters like total distance traveled, maximum velocity, and specific durations of acceleration, deceleration and state of constant velocity. This ensures that we can get to where we want, when we want, consistently in the autonomous mode. It also enables us to make very precise in place turns by running trapezoidal velocity profiles in opposite directions on either side of the robot.

A trapezoidal velocity profile is actually a large array of setpoints based on position for the robot drive. Though called a "velocity profile", the velocity and acceleration parameters are all implicitly defined through the array of setpoints, which each specify a position and duration for which the setpoint must be executed. As soon as the controller completes its motion to the current setpoint in the array, it will proceed to the next point. Once all of these are completed, the controller will stop movement, and has effectively completed the motion profile.





The implementation of motion profiling for our Robot’s Drivetrain was carried out in Java using the Talon SRX’s newly introduced motion profiling capabilities. We used encoders on our drivetrain as the feedback devices. The architecture of our motion profiling system was based of off three software components provided to us. These were:

- Top Level Motion Profile Buffer(provided by WPILib)
- Bottom Level Motion Profile Buffer(Talon SRX)
- Motion Profile Executer(Talon SRX)

To successfully execute a Motion Profile, we implemented a state machine with multithreading which executed the following steps:

- Dynamically generate array of Motion Profile Trajectory Points(TPs) using parameters including total distance, maximum velocity, acceleration times and deceleration times

- Feed all of the TPs into the top level buffer(this buffer can hold more than 1000 TPs)
- Using a thread running every 10 ms, feed the TPs from the Top Level Buffer to the Bottom Level Buffer(This buffer can only hold 128 TPs at a time)
- Execute the TPs by loading them from the Bottom Level Buffer to the Motion Profile Executer. Each point has a duration of 20 ms. The thread loading the TPs into the bottom level buffer is happening simultaneously, at twice the rate of execution to ensure a steady supply.

This motion profiling was ultimately integrated with our Global Positioning system to ensure extremely precise control of our robot in the autonomous mode. This allows us to cross defenses very effectively and consistently, and it also enables us to have an autonomous mode which can shoot in the high goal. Our motion profiling was also used in our vision alignment system to ensure consistent, accurate alignment.

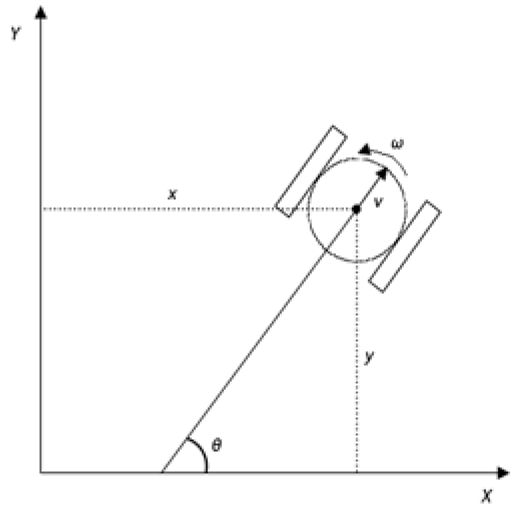
Global Positioning and Orientation with Multiple Sensor Feedback

Because obstacles on the field are arranged in a static manner, it becomes highly advantageous to denote a robot's position with cartesian coordinates. Sensory data, such as motor encoder ticks, ultrasound data, and gyroscope data, can all be used to estimate the robot's cartesian position and rotational orientation on the field since the start of a game match. This not only allows for effective navigation between non-moving obstacles during autonomous routines, but it also supports in vision-guided ball shooting and low-goal shooting. However, because the robot essentially relies on dead-reckoning for position, one important consideration is cumulative errors that occur in the encoder tick system.

1. Using Encoder Ticks as a Base for Global Positioning

On our symmetric robot that employs a tank drive system, quadrature encoders are fixed to both sides of the robot on the motor assembly. This enables the quadrature encoder to relay a certain number of ticks to our motor controller, the Talon SRX. The amount of ticks registered by the quadrature encoder is directly proportional to the linear displacement of the corresponding tread. A conversion constant is used to determine how far, in inches, the tread tracks have displaced linearly, from quadrature encoder data. We previously used a mathematically derived constant that took into account tread diameter, gear ratio, and encoder tick resolution (the number of ticks registered by one rotation of the encoder shaft). However, we found that an empirically defined constant allowed the robot to go to test points much more accurately, Within ±2. Inches.

Given that the robot can calculate the linear displacement on each tread, it can find its orientation (direction angle) and center position. First, the number of ticks are registered on each encoder in an infinitesimally small time frame within a loop. The robot then uses basic trigonometry to determine the change of the robot’s orientation, namely omega (w), as a function of the difference of ticks on each side of the robot.



(http://www.cs.cmu.edu)

Through the net linear displacement of the centerpiece, as a function of the sum of the robot left and right encoder ticks, the robot can calculate the change in its own cartesian position in each time frame. Because the incremental changes in all our variables are so small in each loop, we can use the small sine approximation to find our displacement. Cumulatively, these cartesian contributions to the x-coordinate and y-coordinate of the robot determine its field position.

2. Using the Gyroscope to correct Orientation

Encoders are most accurate when the robot travels straight forward or backward. When performing zero-radius turns however, more accurate sensors can be used to sense orientation changes. On our robot, we use a gyroscope to angle changes from such sharp turns. In general, our robot uses a t-s-t movement system (Turn, Straight, Turn). This scheme breaks up a path from a source node to a destination node on the cartesian grid into a series of alternating straight line movements and turns, to maneuver around objects. The gyroscope comes into play in two different ways. First, whenever a zero-radius turn maneuver is performed, rather than using encoder ticks, we can rely on the gyro sensor to more accurately estimate the robot’s rotational orientation. The second method of implementation is to use the gyroscope to confirm that the robot is not slanting and is thus going straight without angling off. If the robot does bend off, the gyroscope calculates this drift and then reports it to the robot. The robot will then adjust for the drift before continuing with the next commands. This is especially important in the autonomous mode in the game as the robot will be able to independently adjust its orientation based on live changes on the field.

3. Dealing With Rough Terrain and Cumulative Errors in Encoder Ticks

The encoder tick and gyroscope system is fairly reliable to find our position and orientation, given that the tracks are in constant contact with a level floor for the duration of the robot's performance. One obvious problem with relying only on encoder ticks is that once it travels over a game defense, such as the Rough Terrain, tread contact with the floor is frequently interrupted. This disorientation results in a loss of information about linear displacement, and so our position on the field-coordinate system becomes inaccurate. In order to correct our position data, we propose to use the ultrasonic sensors to readjust our global positioning coordinates. The robot would employ four ultrasonic sensors to calculate the orthogonal distance from fixed field elements, such as walls, and defense shields at advantageous situations, such as right after a defense cross. We can thus manipulate the coordinates for the fixed objects with our calculated distances in order to compute our position on the field. These distances can be utilized to calculate our coordinates by By using these distances to provide rough estimates of our coordinates on the field, we can reliably use the global positioning system regardless of any accuracy issues in the encoders.

Vision Processing for High Goal Shooting

Our vision processing system allows autonomous shooting in the high goal, as well as automated alignment and shooting into the high goal during the tele-operated mode of the game. There are numerous steps in our vision processing, and in conjunction, these steps give us the capability to align to the high goal and estimate how far away our robot is from the high goal.

Step 1: GRIP Vision Filtering with Raspberry Pi

We used the GRIP GUI for OpenCV introduced by WPILib for this season. We use an Axis Camera with 3 white ring lights, and we used such high power lights to ensure that we could see the high goal from the very edge of the defenses, about 14 feet away from the high goal. We then created a custom GRIP filter using various images taken from different angles and distances, to ensure that our robot could see and recognize the retroreflective tape around the high goal. After all filtering was complete, the tape emerged as a U-shaped white contour which could clearly be seen from as far as 20 feet away.

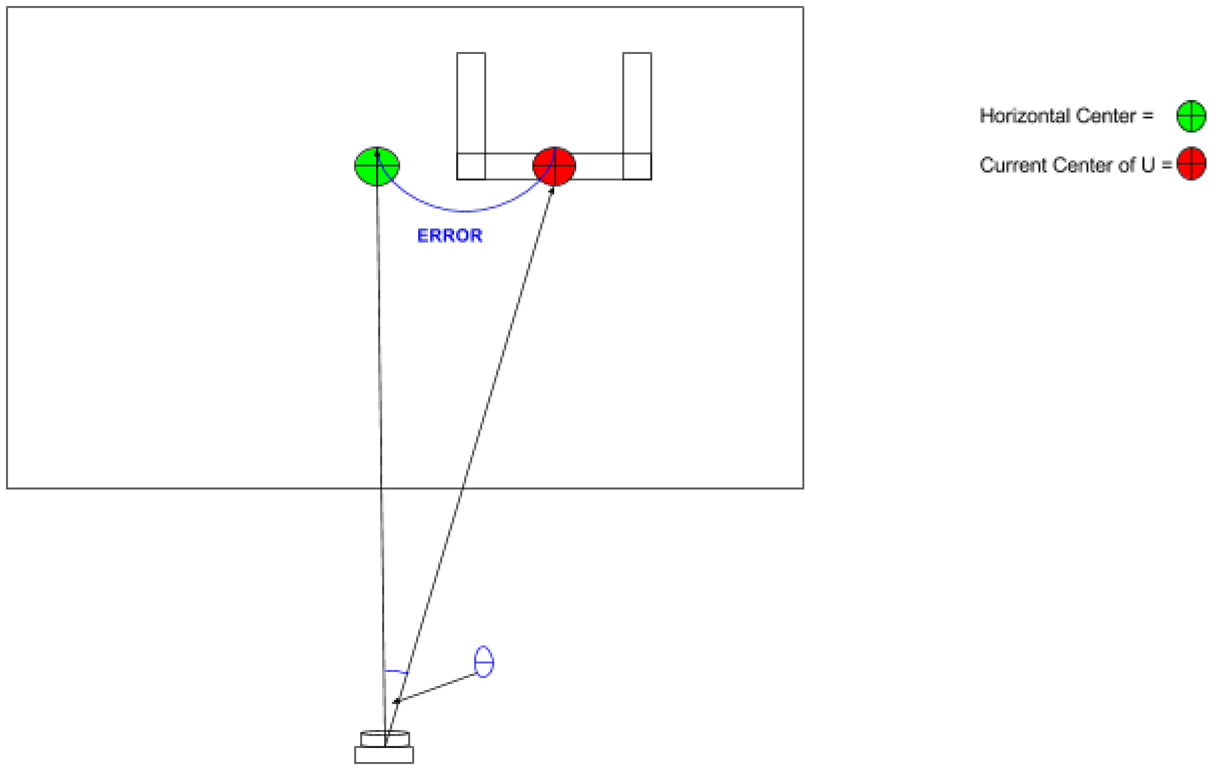
The camera settings required to get such accurate recognition of the goal retroreflective tape needed much bandwidth. They included running the maximum resolution on the Axis Camera, an 800 pixel by 600 pixel image, and running the camera at a rate of 25 frames per second. As a result, we realized that it was not feasible to run this processing solely on the RoboRIO or our team's computer. So, we decided to offload the processing to a Raspberry Pi 2. A script was written which ran our customized GRIP filter on the image from the Axis Camera, and then determined the Center X and Y coordinates of the U-shaped contour(in pixels) seen after all of the filtering was completed. The script then published this data to the WPILib Network Tables, where our code could access the data and use it for the following steps.

Step 2: Vision Alignment to Goal

After getting the required data from the Raspberry Pi, we could then use it to align our robot so that it could shoot perfectly into the high goal. To do this, we used a rather interesting algorithm, specified below:

- Read Center X from Network Tables
- Error = Center X - 400
- Using DEGREES_PER_PIXEL ratio, convert Error → Degrees Error
- Use Trapezoidal Velocity Profile to turn robot in place for required Degrees Error

The algorithm detailed above took advantage of our precise trapezoidal velocity profiles to ensure accurate vision alignment. What we basically did is measured the horizontal field of view of the camera in degrees. We knew that this field of view was also 800 pixels wide, so we then mapped degrees to pixel through the DEGREES_PER_PIXEL ratio. This ratio basically tells us that if the Center X is p pixels away from 400, then the robot drive is θ degrees away from perfect central alignment with the goal(In this case, DEGREES_PER_PIXEL = θ/p).
After determining this angle θ , the offset of the drive, we just used our trapezoidal velocity profiling to make a turn in place for that angle θ , which ensures proper alignment. The diagram below displays the logic behind the algorithm:

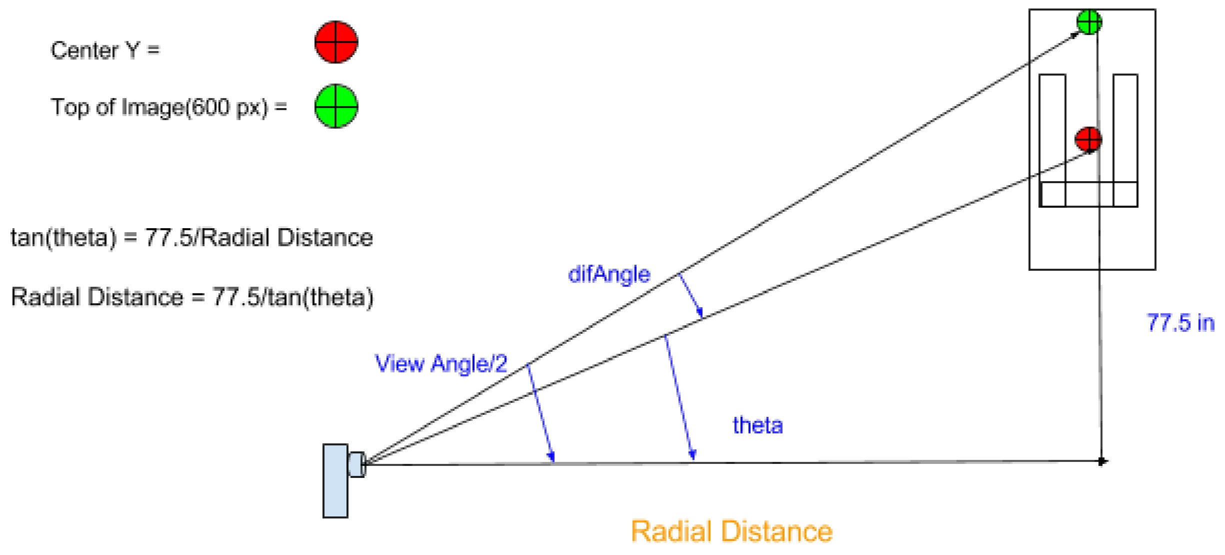


Step 3: Radial Distance Estimation Using Vision

Once alignment is complete, we used another algorithm to estimate the distance between the center and the goal and the camera. We did this using the Center Y coordinate provided to us by the Raspberry Pi. Center Y is taken with the origin of reference being the top edge of the image. So first, we determined the Vertical View Angle of the camera. We once again related this view angle to pixels, since the full vertical view would be 600 pixels. It was also important for us to know the difference in height between the center of our camera and the center of the goal, which is 77.5 inches(use explained later). We then used the following algorithm to determine how far away we were:

- Read Center Y from Network Tables
- DifAngle = Center Y * View Angle/600
- Theta = View Angle/2 - DifAngle
- RadialDistance = 77.5/tan(Theta)

The diagram below explains the logic behind the algorithm:



In conjunction, both the vision alignment to the high goal and the radial distance estimation using vision have proved to be extremely accurate. This allows us to automatically align to the goal, raise our shooter to the appropriate angle based on distance, and shoot, in both Teleoperated and Autonomous modes.

Multipurpose Articulating Arm

Using an articulating arm on the front of the robot allowed us to both intake and shoot the ball with the the same mechanism. This reduced the amount of extraneous code that had to be written. Additionally we were able to use both encoders and potentiometers to accurately control the speed at which we shoot as well as the angle.

Using PID speed control with encoders when shooting

In order to be able to shoot the ball accurately we had to be able to precisely control the speed of it. There were two aspects of the ball's travel in particular that we controlled: direction and distance. Originally we ran the wheels in percentVbus mode, which powered them only using a certain voltage amount. While voltage mode allowed us to shoot far enough, the problem came with the latter. Powering the motors with the same voltage did not ensure that they would spin at the same rate, but using encoders would have. We mounted the encoders to the shooter wheels and PID tuned them so that we had precise control over them no matter what speed we wanted them to spin at. Having both motors spin at the same speed in encoder ticks per 100ms reduced the amount of variation in spin.

References

Brooks, Rodney A. “A Robust Layered Control System for a Mobile Robot” MIT Artificial Intelligence Laboratory September 1985

L. Feng , J. Borenstein , and H. R. Everett “‘Where am I?’ Sensors and Methods for Autonomous Mobile Robot Positioning” University of Michigan December 1994