The Programming Bible

Team 564

Last updated 1/21/16

Here is a link to videos to help with understanding all of this

https://www.thenewboston.com/videos.php?cat=31&video=17966

PRIMITIVE DATA TYPES (TO DEFINE A VARIABLE)

- int – data type for integers (-1, 2, 33)
- double- data type for decimals (1.0, 1.5, -2.59)
- String – data type for text (“yolo”, ”a”)
- char - (no one uses this) data type for a single letter (‘a’)
- boolean - data type for “yes” or “no” (true, false)

OPERATORS (DOING STUFF WITH VARIABLES)

Simple operators

- = : Equals (i = 2)
- + : Add (2 + 2)
- - : Subtract (2 – 2)
- * : Multiply (2 * 2)
- / : divide (2 / 2)
- % :Modulus (The remainder. EX: the modulus of 4/2 is 0 and 3/2 is 1)
- [ ] = array- a list starting at 0

  ~1st term(0) is called the parent~

The Unary Operators (This will be important when doing real world math with programming)

- +X : positive-ify number
- -X : negative-ify number
- X++ : add 1 to X
- X-- : subtract 1 from X
- !X : Meaning not, used with a boolean (false == !true, true == !false)

ADVANCED MATH OPERATORS

CHANGING VALUE OF A VARIABLE THE LAZY WAY: X <math operator>= #

- X += 2 : add 2 to X **(same as X = X + 2)**
- X -= 2 : subtract 2 from X
- X *= 2 : multiply X by 2
- X /= 2 : divide X by 2

~Equality and Relational Operators

- == : equal to (not the same as " = ") (EX: x == y) (Is used to test whether 2 primitive variables are equal to each other)
- != : Not equal to (2 != 3, Christmas != thanksGiving)
- > : greater than
- < : less than
- >= : greater than or equal to (don't be a silly willy by doing "=>")
- <= : less than or equal to

~Conditional Operators (True and false stuff)

- && : and (EX: Apple && Orange , apple and orange are true. Both must be true to work)
- || : or (EX: !snowing || raining , it is not snowing or is raining, only one of them needs to be true to work

EXPRESSIONS, STATEMENTS,

Display HelloWorld in console:

```
System.out.println("HelloWorld");
<object>.<method>(<parameters>)
```

- System.out is the name of the object that is used for output.
- println is the method used to output text at the very bottom of the console window.
- "HelloWorld" is the parameter being given to println. println only has one parameter: the String of text to be displayed.
- Semicolons ; mark the end of each statement or sentence in a program
- Object oriented programs accomplish tasks by sending messages to objects, a System.out object responds to a println message by printing a string of characters in the terminal window.
- the period (.) between object and the method is called a method selector. the period between System and out is not a method selector, it is part of the obj

Import Statements

~Importing a utility and initializing it.

1. Create an import statement above “public class”. The General form of an import statement consist of the overall name of the package, the name of a subsection within the package and the name of a particular class in the subsection. import is the command to create an import statement

Ex. import java.util.Scanner; (Java is the package, util is the subsection, Scanner is the class),
mport is the command to create an import statement

2. initiate the object from the imported class.
   SomeClass someObject = new SomeClass(some parameter);
   Ex. Scanner reader = new Scanner(System.in); (This initiates a scanner that is called “reader”)
3. Create a method

Some utilities require and I/O (Input, output) data such as a Scanner. For example, A programmer creates a piece of code that asks the user to input an integer for a variable using a scanner utility. it would look something like this
int a = 0;
System.out.println(“Please specify an input”);
a = reader.nextInt(); (nextInt() returns the first integer in the input line, basically you use this to assign a variable a value) (BEWARE USE THE CORRECT METHOD FOR THE CORRECT VARIABLE, nextDouble() does not work with an integer variable!!!)

Logic, Booleans, and Loops

- If and else statements-An if and else statement is a simple conditional expression.
- if (a > b) System.out.println("a ackbar");
- else if (b > 3) System.out.println("a <= b, b > 3");
- else if (b >= 2) System.out.println("a <= b, b >= 2");
- else System.out.println("a < b, b < 2");

```
If(condition){
    statement; //Execute these statements if condition is true
}
else{
    statement; //Execute this statement if the condition if false
}
```

- While-A while contains a condition that creates certain parameter such as x < 20, and under that condition are statements that usually define the a new condition for the while statement

EX.

```
int x = 0;
while (x < 20){ // until x is greater than 20, the code will keep looping
x++; // increment x (add 1), this will become the new x, and go back to the
    condition in the while statement.
}
```

The result will be that x is equal to 20, because eventually the condition in the while statement will become false, and will terminate.

- Loops-A loop is a code segment that repeats itself continuously as long as a certain condition remains true, once the condition is false, the repetition will stop.

Ex-

```
int counter=0
 while(counter < 10) {
        system.out.println(counter);
        counter ++;
}
```

Result

0
1
2
3
4
5
6
7
8
9

Using Multiple Classes

Template:

- <name>Object = new <classname>(<parameters if any>);
- <name>Object.<method>(<parameters>);

- name.Object = name that you will be using to call forth the class
- classname= the name of the class you will be calling
- method= the method from the class you want to use, remember to separate the <name>.Object from the method using a "."

Ex. tunaObject = new tuna(); ⇇ Specify your method
tunaObject.simplemessage(); ⇇ use your method

MULTIPLE CLASSES

· **CLASSES SERVERS AND CLIENTS**
· **CLASS:** a class is a software package or template that describes the characteristics of similar objects
· **CLIENT THE BOSS:**A computer computation object that receives a service from another computational object
· **SERVER WORKER:**A computational object that provides a service to another computational object
· **OBJECT:**A collection of data and operations in which the data can be accessed and modified only by means of the operations
· **CONSTRUCTOR:** A method that is run when an object is instantiated usually to initialize that object variable
· **METHOD:** A chunk of code that can be treated as a unit and invoked by name
· **METHOD HEADING:** The portion of a method Implementation containing the function's name, parameter declarations and return type
· **PARAMETER:** A variable or expression contained in a method and call and passed to another method
· **STRING METHOD :** Swbat: use the string method
· **CHARAT(ANINDEX):** returns char Example: myStr.charAt(4); returns the character at the position anindex. Remember that the first character is at position 0. An execution is thrown (i.e an error is generated) if anIndex is out of range (i.e. does not indicate a valid position with myStr)
· **COMPARETO(ASTRING):** returns int Example I = myStr.compareTi("abc"); compares two strings alphabetically. Return 0 if myStr equals aString a value

less than 0 if myStr string is alphabetically less than aString and a value greater than 0 if myStr string is alphabetically greater than aString

· **EQUALS(ASTRING):** Returns boolean ExAmple:boolean = myStr.equals("abc"); Returns true if myStr equals aString; else returns false. Because of implementation peculiarities in java, never test for equality like this: myStr == aString.

· **EQUALSIGNORECASE(ASTRING):** Returns Boolean Similar to equals but ignores but ignores case ignores case during the comparison

· **INDEXOF(ACHARACTER):** Returns int Example: I = myStr.indexOf('x'); Returns the index within myStr of the first occurrence of a Character or -1 if aCharactrer is absent

·**INDEXOF(ACHARACTE, BEGININDEX):** returns int Example: I = myStr.indexOf('z', 6); Similar to the preceding method except the search starts at position beginIndex rather than at the beginning of myStr. An exception is thrown (i.e. an error is generated) if beginning

· **INDEXODF(ASUBSTRING):** Returns int Example: I = myStr.indexOF("abc"); Returns the index within myStr of the first occurrence of aSubString or -1 if aSubString is absent

· ***LENGTH:** Returns int Example: I = myStr.length(); Returns the length of myStr.

· **REPLACE(OLDCHAR, NEWCHAR):** Returns string Example: Str = myStr.replace('z', 'b'); Returns a new string resulting from replacing occuerences of oldchar in myStr with newChar. myStr is not.

·***SUBSTRING(BEGININDEX):** returns string Example str = myStr.sugbString(6); Returns new string that is a substring of myStr. The substring begins at location beginindex and extends to the end of myStr an exception is thrown(i.e, ab error is generated beginindex is out of range(i.e, does not indicate a valid position within myStr)

· ***SUBSTRING(BEGININDEX, ENDINDEX):** Returns string Example: str = myStr.subString(4,9); Similar to the preceding method except the substring extends ton location endindex -1 rather than to the end of myStr

· **TOLOWERCASE():**Returns string Example str = myStr.LowerCase(); Str is the same as myStr except that all letters have been converted to lowercase myStr is not changed

· **TOUPPERCASE():**Returns string Example str = myStr.LowerCase(); Str is the same as myStr except that all letters have been converted to uppercase myStr is not changed

· **TRIM():** Returns string Example str = myStr.trimm(); Str is the same as myStr except the leading and trailing spaces if any are absent myStr is not

changed
Sensor Code

Gyro:

void	***initGyro ()*** *Initialize the gyro.*
	Gyro (int channel) ***Gyro*** *constructor using the channel number.*
	Gyro (AnalogInput channel) ***Gyro*** *constructor with a precreated analog channel object.*
void	***reset ()*** *Reset the gyro.*
doube	***getAngle ()*** *Return the actual angle in degrees that the robot is currently facing.*
void	***setSensitivity (double voltsPerDegreePerSecond)*** *Set the gyro sensitivity.*

Joystick:

doube	***getY (Hand hand)***
	Get the Y value of the joystick.

int	***getButtonCount ()***
	For the current joystick, return the number of buttons.

boolen	***getButton (ButtonType button)***
	Get buttons based on an enumerated type.

doube	***getDirectionDegrees ()***
	Get the direction of the vector formed by the joystick and its origin in degrees.

int	***getAxisChannel (AxisType axis)***
	Get the channel currently associated with the specified axis.

void	***setAxisChannel (AxisType axis, int channel)***
	Set the channel associated with a specified axis.

void	***setOutput (int outputNumber, boolean value)***
	Set a single HID output value for the joystick

Compressor:

	Compressor (int pcmId) Create an instance of the Compressor.
	Compressor () Create an instance of the Compressor Makes a new instance of the compressor using the default PCM ID (0).
void	**start ()** Start the compressor running in closed loop control mode Use the method in cases where you would like to manually stop and start the compressor for applications such as conserving battery or making sure that the compressor motor doesn't start during critical operations.
void	**stop ()** Stop the compressor from running in closed loop control mode.

Talons:

	Talon (final int channel) Constructor for a Talon (original or Talon SR)
void	**set (double speed, byte syncGroup)** Set the PWM value.

Solenoids:

Solenoid (final int channel)

Constructor using the default PCM ID (0)

Solenoid (final int moduleNumber, final int channel)

Constructor.

void	**set (boolean on)** Set the value of a solenoid

Victors:

Victor (final int channel)

Constructor.

void	**set (double speed, byte syncGroup)** Set the PWM value.

void	**set (double speed)** Set the PWM value

void	**stopMotor ()**

Stop the motor associated with this PWM object.

CameraServer:

void	***setImage (Image image)***
	Manually change the image that is served by the MJPEG stream.

void	***startAutomaticCapture ()***
	Start automatically capturing images to send to the dashboard.

void	***startAutomaticCapture (String cameraName)***
	Start automatically capturing images to send to the dashboard

Sensor code for Autonomous ***(This section will be updated according to the game)***

Variables:

- Jag1, Jag2, Jag3, Jag4 = wheels
-
-
-
-
-
-

Codes:

VD.set

- val.set(#); <= “#” can be set from -1 to 1 like percentages -100% to 100%
- Jag1.set(#); = “#” same as ⇧above⇧
- Timer.delay(#);= “#” set in seconds. this is time delay before continuing to next codes
- Jag#.StopMotor(); = stops motor
-
-
-
-
-
-

☆ ***Reminders*** ☆

Everything will keep running as long as it is set on true (except Jaguars where you use Jag#.StopMotor();)

DO NOT MAKE ROBOT USE CONTRADICTING CODES SIMULTANEOUSLY

Timer delay lets the codes above it run before starting the codes bel

Autonomous Code

~Getingstrated.java code~

```
package org.usfirst.frc.team564.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.RobotDrive;
import edu.wpi.first.wpilibj.livewindow.LiveWindow;

/**
* The VM is configured to automatically run this class, and to call the
* functions corresponding to each mode, as described in the IterativeRobot
* documentation. If you change the name of this class or the package after
* creating this project, you must also update the manifest file in the resource
* directory.
*/
public class Robot extends IterativeRobot {
	RobotDrive myRobot;
	Joystick stick;
	int autoLoopCounter;

  /**
   * This function is run when the robot is first started up and should be
   * used for any initialization code.
   */
  public void robotInit() {
	myRobot = new RobotDrive(0,1);
	stick = new Joystick(0);
  }

  /**
   * This function is run once each time the robot enters autonomous mode
   */
  public void autonomousInit() {
	autoLoopCounter = 0;
```

```
    }

    /**
     * This function is called periodically during autonomous
     */
    public void autonomousPeriodic() {
        if(autoLoopCounter < 100) //Check if we've completed 100 loops (approximately 2
seconds)
                {
                        myRobot.drive(-0.5, 0.0);       // drive forwards half speed
                        autoLoopCounter++;
                        } else {
                        myRobot.drive(0.0, 0.0);        // stop robot
                }
    }

    /**
     * This function is called once each time the robot enters tele-operated mode
     */
    public void teleopInit(){
    }

    /**
     * This function is called periodically during operator control
     */
    public void teleopPeriodic() {
        myRobot.arcadeDrive(stick);
    }


    /**
     * This function is called periodically during test mode
     */
    public void testPeriodic() {
        LiveWindow.run();
    }

}
```

~Gyro.Java code~

```
package org.usfirst.frc.team564.robot;

import edu.wpi.first.wpilibj.CANTalon;
import edu.wpi.first.wpilibj.AnalogGyro;
import edu.wpi.first.wpilibj.SampleRobot;
import edu.wpi.first.wpilibj.RobotDrive;
import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.smartdashboard.*;

/**
* This is a sample program to demonstrate how to use a gyro sensor to make a robot drive
* straight. This program uses a joystick to drive forwards and backwards while the gyro
* is used for direction keeping.
*
* WARNING: While it may look like a good choice to use for your code if you're inexperienced,
* don't. Unless you know what you are doing, complex code will be much more difficult under
* this system. Use IterativeRobot or Command-Based instead if you're new.
*/
public class Robot extends SampleRobot {

    final int gyroChannel = 0; //analog input
    final int joystickChannel = 0; //usb number in DriverStation

    //channels for motors
    final int leftMotorChannel = 1;
    final int rightMotorChannel = 0;
    final int leftRearMotorChannel = 3;
    final int rightRearMotorChannel = 2;

    double angleSetpoint = 0.0;
    final double pGain = .006; //propotional turning constant

    //gyro calibration constant, may need to be adjusted;
    //gyro value of 360 is set to correspond to one full revolution
    final double voltsPerDegreePerSecond = .0128;
```

```
RobotDrive myRobot;
AnalogGyro gyro;
Joystick joystick;

public Robot()
{
        //make objects for the drive train, gyro, and joystick
        myRobot = new RobotDrive(new CANTalon(leftMotorChannel), new CANTalon(
                leftRearMotorChannel), new CANTalon(rightMotorChannel),
                new CANTalon(rightRearMotorChannel));
        gyro = new AnalogGyro(gyroChannel);
        joystick = new Joystick(joystickChannel);

}

   /**
    * Runs during autonomous.
    */
   public void autonomous() {

   }

   /**
    * Sets the gyro sensitivity and drives the robot when the joystick is pushed. The
    * motor speed is set from the joystick while the RobotDrive turning value is assigned
    * from the error between the setpoint and the gyro angle.
    */
   public void operatorControl() {
        double turningValue;
        gyro.setSensitivity(voltsPerDegreePerSecond); //calibrates gyro values to equal degrees
      while (isOperatorControl() && isEnabled()) {


        turningValue =  (angleSetpoint - gyro.getAngle())*pGain;
        if(joystick.getY() <= 0)
        {
         //forwards
         myRobot.drive(joystick.getY(), turningValue);
        } else {
```

```
            //backwards
            myRobot.drive(joystick.getY(), -turningValue);
           }
        }
    }

    /**
     * Runs during test mode.
     */
    public void test(){

    }
}
```

~Tankdrive.java code~

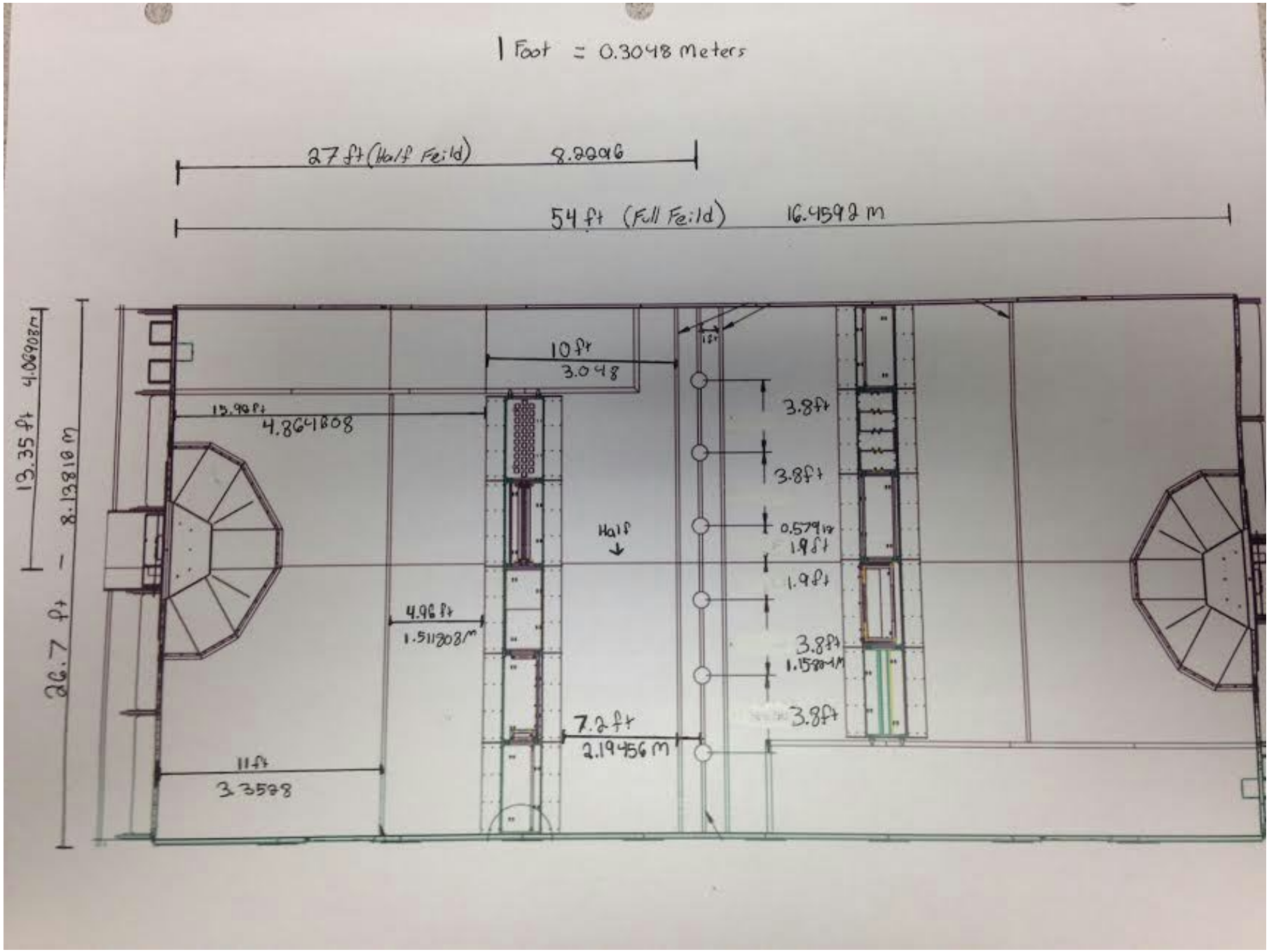
```
package org.usfirst.frc.team564.robot;

import edu.wpi.first.wpilibj.SampleRobot;
import edu.wpi.first.wpilibj.RobotDrive;
import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.Timer;
/**
* This is a demo program showing the use of the RobotDrive class, specifically it
* contains the code necessary to operate a robot with tank drive.
*
* The VM is configured to automatically run this class, and to call the
* functions corresponding to each mode, as described in the SampleRobot
* documentation. If you change the name of this class or the package after
* creating this project, you must also update the manifest file in the resource
* directory.
* WARNING: While it may look like a good choice to use for your code if you're inexperienced,
* don't. Unless you know what you are doing, complex code will be much more difficult under
* this system. Use IterativeRobot or Command-Based instead if you're new.
public class Robot extends SampleRobot {
   RobotDrive myRobot;  // class that handles basic drive operations
   Joystick leftStick;  // set to ID 1 in DriverStation
   Joystick rightStick; // set to ID 2 in DriverStation
   public Robot() {
      myRobot = new RobotDrive(0, 1);
      myRobot.setExpiration(0.1);
      leftStick = new Joystick(0);
      rightStick = new Joystick(1);
   }
   /**
    * Runs the motors with tank steering.
    */
   public void operatorControl() {
      myRobot.setSafetyEnabled(true);
      while (isOperatorControl() && isEnabled()) {
          myRobot.tankDrive(leftStick, rightStick);
         Timer.delay(0.005);              // wait for a motor update time
```

Robotics 2016

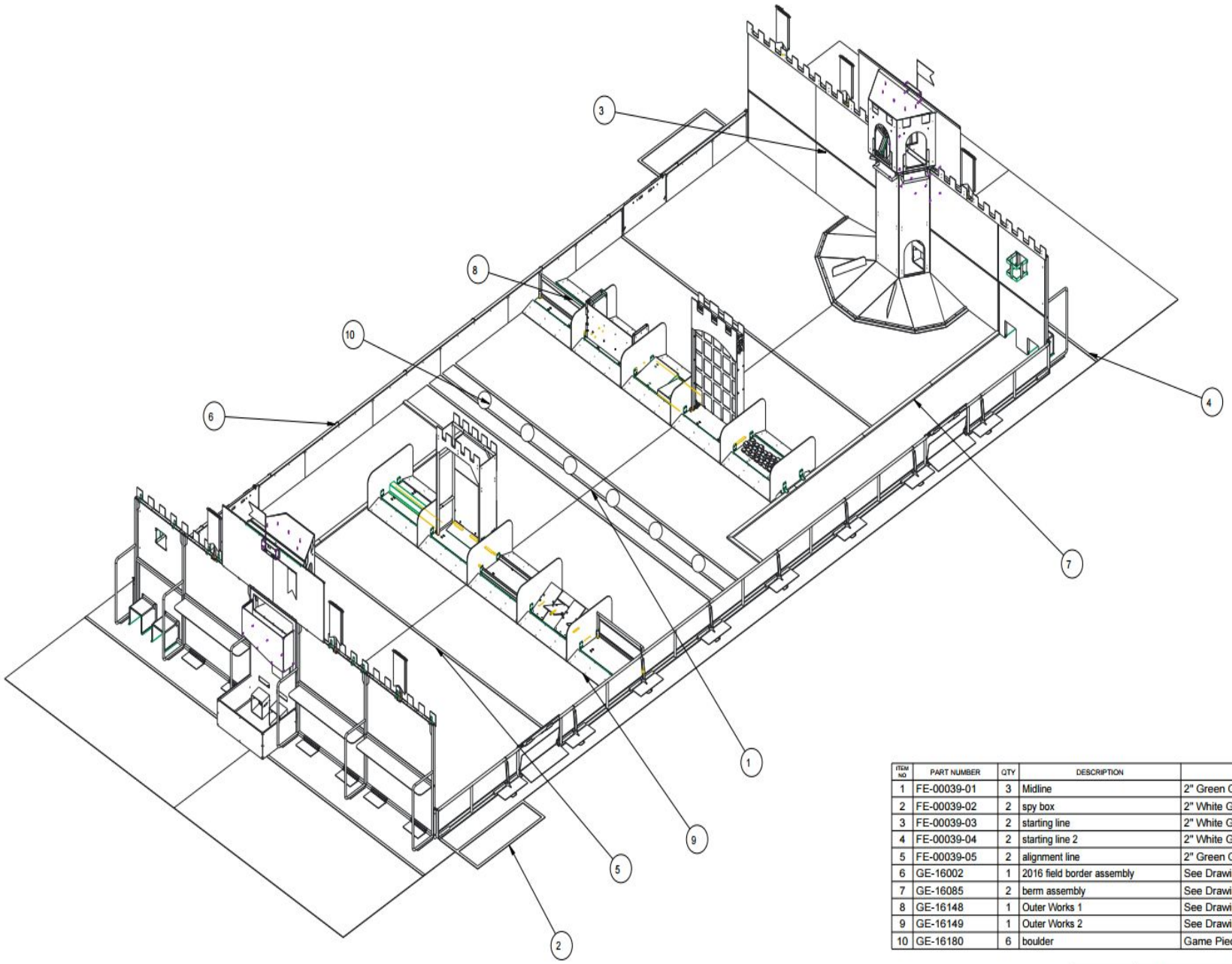
~

Field Drawings And Point Values

(This section will be updated according to the game)


1 Foot = 0.3048 Meters
27 ft (Half Field) 8.2296
54 ft (Full Field) 16.4592 m
13.35 ft 4.069908 m
26.7 ft - 8.13818 m
10 ft
3.048
15.96 ft
4.864808
Half
4.96 ft
1.511808 m
11 ft
3.3528
7.2 ft
2.19456 m
3.8 ft
3.8 ft
0.579 m
1.9 ft
1.9 ft
3.8 ft
1.1582 M
3.8 ft

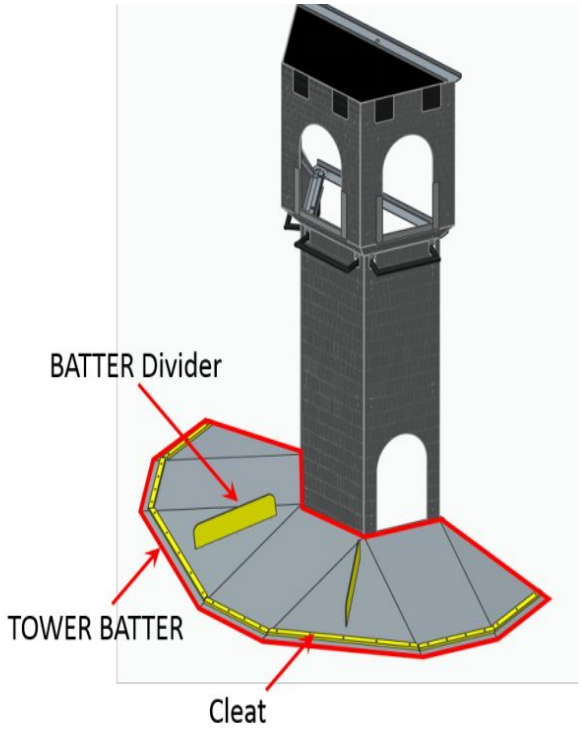
FE-00039 | 2016 FRC Field Assembly

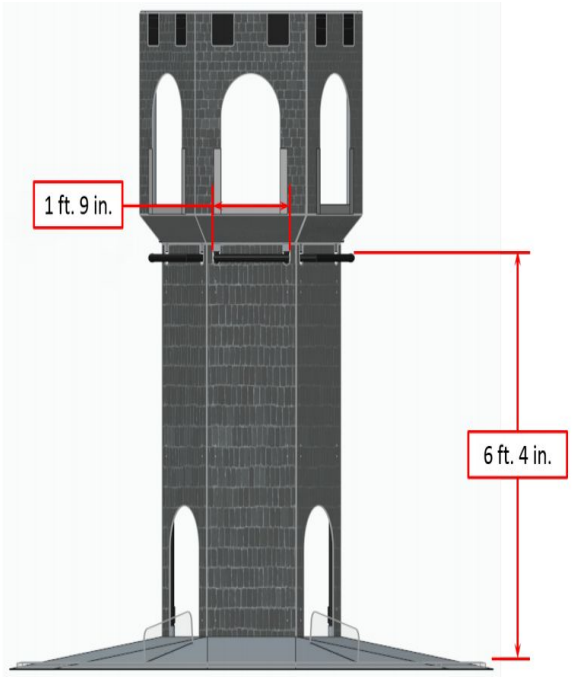


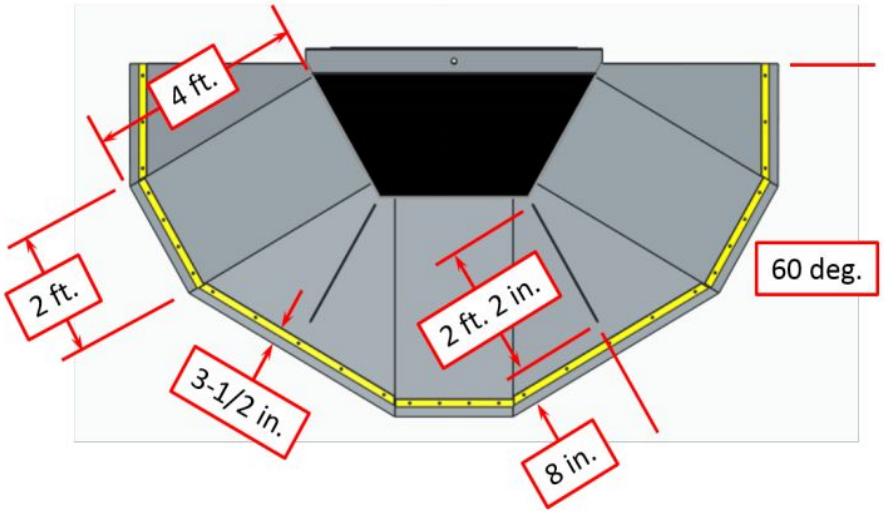
ITEM NO	PART NUMBER	QTY	DESCRIPTION	NOTES
1	FE-00039-01	3	Midline	2" Green Gaffer's Tape
2	FE-00039-02	2	spy box	2" White Gaffer's Tape
3	FE-00039-03	2	starting line	2" White Gaffer's Tape
4	FE-00039-04	2	starting line 2	2" White Gaffer's Tape
5	FE-00039-05	2	alignment line	2" Green Gaffer's Tape
6	GE-16002	1	2016 field border assembly	See Drawing
7	GE-16085	2	berm assembly	See Drawing
8	GE-16148	1	Outer Works 1	See Drawing
9	GE-16149	1	Outer Works 2	See Drawing
10	GE-16180	6	boulder	Game Piece

FIRST

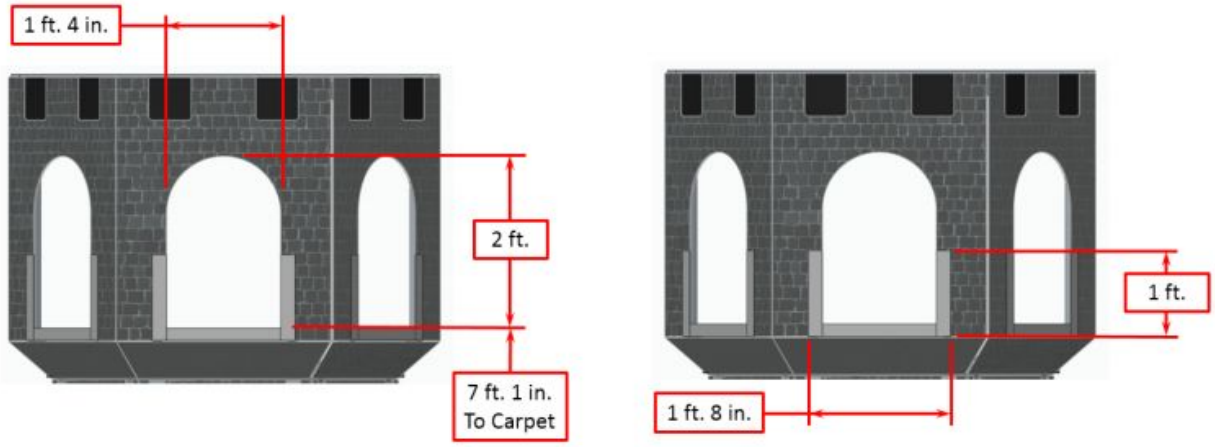
DRAWN BY | DATE
CHECKED BY | DATE
APPROVED BY | DATE
SCALE 1:92 | SHEET 3


BATTER Divider
TOWER BATTER
Cleat


1 ft. 9 in.
6 ft. 4 in.


4 ft.
2 ft.
3-1/2 in.
2 ft. 2 in.
8 in.
60 deg.

High Goal



Low Goal

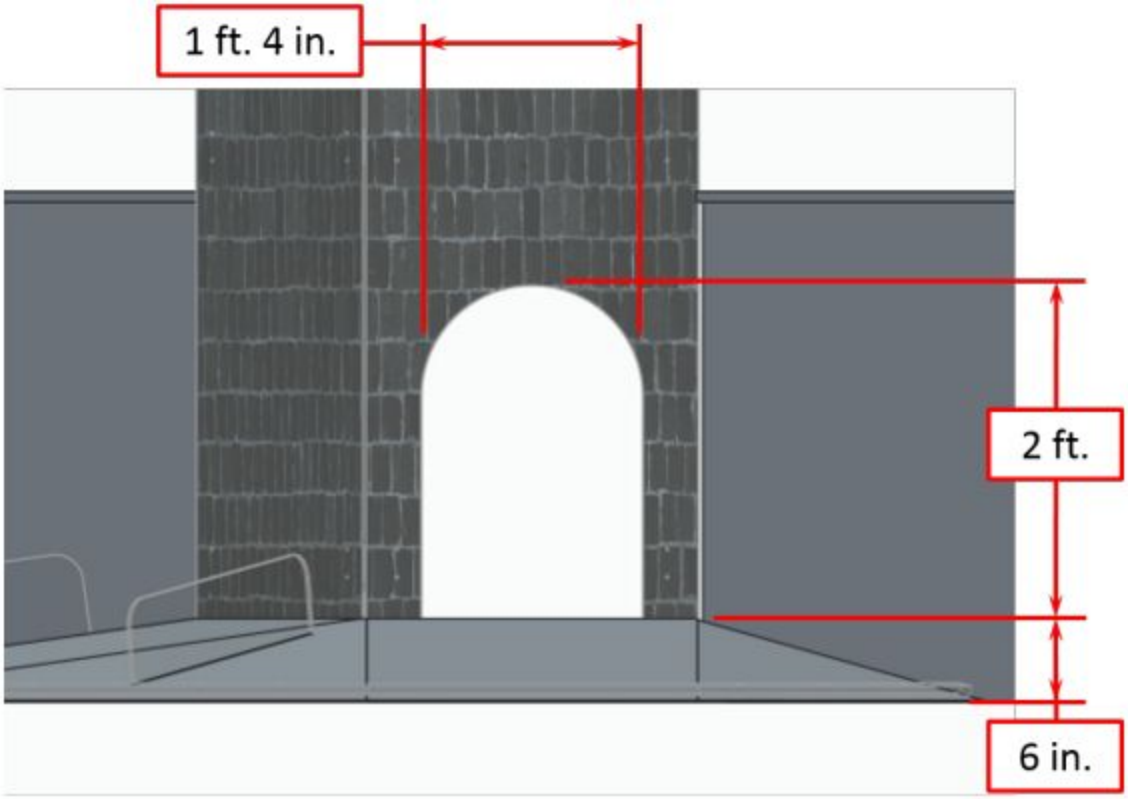


Table 1-1: Auto Point Values

Action	Value
Reaching a defense	2
Crossing a defense	10
Boulder in a low tower goal	5
Boulder in a high tower goal	10

Table 1-2: Teleop Point Values

Action	Value
Crossing a defense	5
Boulder in a low tower goal	2
Boulder in a high tower goal	5
Challenging the tower (per Robot)	5
Scaling the tower (per Robot)	15