public Robot()

Note: The Robot class will take care of getting the UI/Field values from the Radio

Variables:

int[] scores; // stores all teams' scores

int winningScore; // stores score to win Note: may note be needed

bool canMove; // stores if robot can move
Note: if true, Robot class will not allow any actuator change.

bool isAutonomous; // stores if robot should be autonomous
Note: if true, Robot class will not allow user change.
Note: definition of auton could change to "limited usage"

int[] UIAnalogVals; // stores interface's analog values
Note: the interface values are user-input (like joystick)

int[] UIDigitalVals; // stores interface's digital values
Note: the interface values are user-input (like button)

int[] FieldAnalogVals; // stores field's analog values
Note: the field values are user-input (like score, etc.)

int[] FieldDigitalVals; // stores field's digital values
Note: the fields values are user-input (like other info about game)

MotorControllers[] motorControllers; // stores the motor controllers

private ServoControllers[] servoControllers; // stores the servo controllers

private Sensor[] sensors; // stores the sensors
Note: the Robot class won't control the sensors, but the sensors won't worku unless they're connected to the Radio

private Radio radio; // stores instance of Radio object

Methods:

private void run() // uses monitors to safely update motor/servo controllers' states and updates UI/Field values by asking Radio
Note: This is a separate, constantly running thread.

public int[] Scores() // returns all teams' scores

public int WinningScore() // returns score to win Note: may note be needed

public bool CanMove() // returns true if robot can move
Note: if true, Robot class will not allow any actuator change.

public bool IsAutonomous() // returns true if robot should be autonomous
Note: if true, Robot class will not allow user change.
Note: definition of auton could change to "limited usage"

public int[] UIAnalogVals() // returns interface's analog values
Note: the interface values are user-input (like joystick)

public int[] UIDigitalVals() // returns interface's digital values
Note: the interface values are user-input (like button)

public int[] FieldAnalogVals() // returns field's analog values
Note: the field values are user-input (like score, etc.)

public int[] FieldDigitalVals() // returns field's digital values
Note: the fields values are user-input (like other info about game)

public Radio(Robot robot, String port, String sixtyFourID, String sixteenID)

Note: The only features of this class listed right now are the ones the Robot class can access. The rest of this class functions to communicate with the Field/UI Radios to update these variables. The method to write to the UI Radio will come soon.

Variables:

int[] UIAnalogVals; // stores interface's analog values
Note: the interface values are user-input (like joystick)

int[] UIDigitalVals; // stores interface's digital values
Note: the interface values are user-input (like button)

int[] FieldAnalogVals; // stores field's analog values
Note: the field values are user-input (like score, etc.)

int[] FieldDigitalVals; // stores field's digital values
Note: the fields values are user-input (like other info about game)

SimpleMotorController Class(Robot robot, String port, int device number)

Note: If daisychained with an already connected SimpleMotorController, pass the port name of the already existing one.

Properties:

double speed; // accepts between -100, full reverse, to 100, full forward.

double brake; // accepts between 0, coasting, to 10, full braking.

double maxSpeedForward; // accepts 0 to 100

double maxSpeedBackward; // accepts -100 to 0

double maxAccelerationForward; // accepts 0 to 100

double maxAccelerationBackward; // accepts -100 to 0

Methods:

public void updateActuators() // will physically update brake, acceleration, speed
Note: The Robot class should use this in the run thread to update actuators's physical movements.

public void killActuators() // will stop all actuators from functioning until reviveActuators called.
Note: The Robot class should use this when canMove == false;

public void reviveActuators() // will return actuators to functioning state
Note: The Robot class should use this when canMove == true;
Note: This will do nothing if the actuators were never "killed"

MicroMaestro Class(Robot robot, String port, int device number)

Note: If daisychained with an already connected MicroMaestro, pass the port name of the already existing one.
Note: Each property is an array of 6 elements, corresponding to the 6 different channels that servos could be connected on.
Note: defaults occur during initialization of instances

Properties:

double[6] minRotation; // assumed in degrees, defaults to 0 degrees

double[6] maxRotation; // assumed in degrees, defaults to 180 degrees

double[6] targets; // accepts from minRotation to maxRotation

double[6] speeds; // changes how fast servo rotates, accepts 0 to 100
Note: defaults to 50

double[6] accelerations; // changes how fast servo accelerates, accepts 0 to 100
Note: defaults to 50

Methods:

public void rotate(int channel, double degrees) // The argument degrees can be ±. If the rotation moves servo beyond rotation range, the servo will just turn to as extreme of a value it can.
Note: This method is primarily for the students' code's use.

public void updateActuators() // will physically update target, speed, acceleration
Note: The Robot class should use this in the run thread to update actuators's physical movements.

public void killActuators() // will stop all actuators from functioning until reviveActuators called.
Note: The Robot class should use this when canMove == false;

public void reviveActuators() // will return actuators to functioning state
Note: The Robot class should use this when canMove == true;
Note: This will do nothing if the actuators were never "killed"