9228B Tower Takeover Code

2.1.4

Generated by Doxygen 1.8.14

Contents

1 File Index

1.1 File List

Here is a list of all files with brief descriptions:

2 File Documentation

2.1 include/auton.h File Reference

Enumerations

- enum Color : bool { Color::BLUE = false, Color::RED = true }
- enum Side : bool { Side::LEFT = false, Side::RIGHT = true }

Functions

- bool moveForward (double inches, double speed, bool blocking)
- bool pivotClockwise (float degrees, bool blocking)
- bool pivotCounterClockwise (float degrees, bool blocking)
- void auton (Side side, Color color)

  the autonomous switcher
- bool autonStart ()
- void autonBlueLeft ()
- void autonBlueRight ()
- void autonRedLeft ()
- void autonRedRight ()
- void badAuton (Side side, Color color)
- void badAutonBlueLeft ()
- void badAutonBlueRight ()
- void badAutonRedLeft ()
- void badAutonRedRight ()

2.1.1 Enumeration Type Documentation

2.1.1.1 Color

```
enum Color :  bool  [strong]
```

Declares the possible autonomous colors for the autonomous switcher

Author

Michael Baraty

Date

11/9/2019

Enumerator

BLUE	
RED	

Definition at line 38 of file auton.h.

```
00038                  : bool {
00039   BLUE = false,
00040   RED = true
00041 };
```

2.1.1.2 Side

```
enum Side :  bool  [strong]
```

Declares the possible autonomous starting side for the autonomous switcher

Author

Michael Baraty

Date

11/9/2019

Enumerator

LEFT	
RIGHT	

Definition at line 48 of file auton.h.

```
00048                 : bool {
00049   LEFT = false,
00050   RIGHT = true
00051 };
```

2.1.2 Function Documentation

2.1.2.1 auton()

```
void auton (
            Side side,
            Color color )
```

the autonomous switcher

Initiates the specified autonomous routine

Parameters

side	The side in relastion to the zone th erobot is going ot be near
color	The color the robot is starting in

Author

Michael Baraty

Date

11/9/2019

Author

Michael Baraty

Definition at line 64 of file auton.cpp.

```
00064                                   {
00065   autonStart();
00066
00067   if(color == Color::BLUE && side == Side::LEFT)
00068     autonBlueLeft();
00069   else if (color == Color::BLUE && side == Side::RIGHT)
00070     autonBlueRight();
00071   else if (color == Color::RED && side == Side::LEFT)
00072     autonRedLeft();
00073   else if (color == Color::RED && side == Side::RIGHT)
00074     autonRedRight();
00075 }
```

2.1.2.2 autonBlueLeft()

```
void autonBlueLeft ( )
```

Initiates blue left autonomous routine

Author

Michael Baraty

Date

11/9/2019

Definition at line 78 of file auton.cpp.

```
00078                     {
00079   MOTOR_INTAKE_A.startRotateFor(7, rotationUnits::rev, 100, velocityUnits::pct);
00080   MOTOR_INTAKE_B.startRotateFor(7, rotationUnits::rev, 100, velocityUnits::pct);
00081   moveForward(35, 33);
00082   /*pivotClockwise(190);
00083   moveForward(24);
00084   pivotCounterClockwise(45);*/
00085   moveForward(-24, 60);
00086   pivotCounterClockwise(135);
00087   moveForward(10);
00088   moveForward(5, 30, false);
00089   MOTOR_INTAKE_A.startRotateFor(-.5, rotationUnits::rev, 100, velocityUnits::pct);
00090   MOTOR_INTAKE_B.startRotateFor(-.5, rotationUnits::rev, 100, velocityUnits::pct);
00091   MOTOR_STACK.rotateFor(2, rev);
00092   MOTOR_INTAKE_A.startRotateFor(-20, rotationUnits::rev, 100, velocityUnits::pct);
00093   MOTOR_INTAKE_B.startRotateFor(-20, rotationUnits::rev, 100, velocityUnits::pct);
00094   moveForward(-20, 33);
00095   MOTOR_INTAKE_A.stop();
00096   MOTOR_INTAKE_B.stop();
00097 }
```

2.1.2.3 autonBlueRight()

```
void autonBlueRight ( )
```

Initiates blue right autonomous routine

Author

Michael Baraty

Date

11/9/2019

Definition at line 99 of file auton.cpp.

```
00099                     {
00100   intakeIn();
00101   moveForward(25, 70);
00102   moveForward(12, 40);
00103   vexDelay(1000);
00104   MOTOR_INTAKE_A.stop(hold);
00105   MOTOR_INTAKE_B.stop(hold);
00106   moveForward(-28, 80);
00107   pivotClockwise(130);
00108   moveForward(29, 50);
00109   //MOTOR_INTAKE_A.rotateFor(-.25, rotationUnits::rev, 100, velocityUnits::pct);
00110   //MOTOR_INTAKE_B.rotateFor(-.25, rotationUnits::rev, 100, velocityUnits::pct);
00111   //MOTOR_STACK.startRotateTo(2.9, rotationUnits::rev);
00112   //vexDelay(1500);
00113
00114   MOTOR_INTAKE_A.startRotateFor(-10, rotationUnits::rev, 100, velocityUnits::pct);
00115   MOTOR_INTAKE_B.startRotateFor(-10, rotationUnits::rev, 100, velocityUnits::pct);
00116   pivotClockwise(60, true);
00117
00118   moveForward(-13, 33, true);
00119
00120   MOTOR_STACK.rotateTo(0, rotationUnits::rev, 80, velocityUnits::pct);
00121 }
```

2.1.2.4 autonRedLeft()

```
void autonRedLeft ( )
```

Initiates red left autonomous routine

Author

Michael Baraty

Date

11/9/2019

Definition at line 123 of file auton.cpp.

```
00123                   {
00124   intakeIn();
00125   moveForward(25, 70);
00126   moveForward(12, 40);
00127   vexDelay(1000);
00128   MOTOR_INTAKE_A.stop(hold);
00129   MOTOR_INTAKE_B.stop(hold);
00130   moveForward(-28, 80);
00131   pivotCounterClockwise(130);
00132   moveForward(29, 50);
00133   //MOTOR_INTAKE_A.rotateFor(-.25, rotationUnits::rev, 100, velocityUnits::pct);
00134   //MOTOR_INTAKE_B.rotateFor(-.25, rotationUnits::rev, 100, velocityUnits::pct);
00135   //MOTOR_STACK.startRotateTo(2.9, rotationUnits::rev);
00136   //vexDelay(1500);
00137
00138   MOTOR_INTAKE_A.startRotateFor(-10, rotationUnits::rev, 100, velocityUnits::pct);
00139   MOTOR_INTAKE_B.startRotateFor(-10, rotationUnits::rev, 100, velocityUnits::pct);
00140   pivotCounterClockwise(60, true);
00141
00142   moveForward(-13, 33, true);
00143
00144   MOTOR_STACK.rotateTo(0, rotationUnits::rev, 80, velocityUnits::pct);
00145 }
```

2.1.2.5 autonRedRight()

```
void autonRedRight ( )
```

Initiates red right autonomous routine

Author

Michael Baraty

Date

11/9/2019

Definition at line 147 of file auton.cpp.

```
00147                    {
00148   MOTOR_INTAKE_A.startRotateFor(7, rotationUnits::rev, 100, velocityUnits::pct);
00149   MOTOR_INTAKE_B.startRotateFor(7, rotationUnits::rev, 100, velocityUnits::pct);
00150   moveForward(35, 33);
00151   /*pivotClockwise(190);
00152   moveForward(24);
00153   pivotCounterClockwise(45);*/
00154   moveForward(-24, 60);
00155   pivotClockwise(135);
00156   moveForward(10);
00157   moveForward(5, 30, false);
00158   MOTOR_INTAKE_A.startRotateFor(-.5, rotationUnits::rev, 100, velocityUnits::pct);
00159   MOTOR_INTAKE_B.startRotateFor(-.5, rotationUnits::rev, 100, velocityUnits::pct);
00160   MOTOR_STACK.rotateFor(2, rev);
00161   MOTOR_INTAKE_A.startRotateFor(-20, rotationUnits::rev, 100, velocityUnits::pct);
00162   MOTOR_INTAKE_B.startRotateFor(-20, rotationUnits::rev, 100, velocityUnits::pct);
00163   moveForward(-20, 33);
00164   MOTOR_INTAKE_A.stop();
00165   MOTOR_INTAKE_B.stop();
00166 }
```

2.1.2.6 autonStart()

```
bool autonStart ( )
```

Initiates the autonomous routine to release the mechanisms

Author

Michael Baraty

Date

11/9/2019

1. Move intake up so mechanisms deploy
2. Move intake back down
3. Run intake and drive forward to pick up preload
4. Score cubes

Definition at line 174 of file auton.cpp.

```
00174                   {
00175
00176   MOTOR_INTAKE_A.startRotateFor(directionType::rev, 3, rotationUnits::rev, 100,
      velocityUnits::pct);
00177   MOTOR_INTAKE_B.startRotateFor(directionType::rev, 3, rotationUnits::rev, 100,
      velocityUnits::pct);
00178   MOTOR_ARM.rotateTo(2, rotationUnits::rev, 100, velocityUnits::pct);
00179
00180   MOTOR_STACK.startRotateTo(1.5, rev);
00181
00182   MOTOR_ARM.rotateTo(3.9, rotationUnits::rev, 100, velocityUnits::pct);
00183
00184
00185   moveForward(3);
00186
00187   MOTOR_STACK.startRotateTo(0, rev);
00188
00189   MOTOR_ARM.rotateTo(-.15, rotationUnits::rev, 100, velocityUnits::pct);
00190   MOTOR_ARM.stop(hold);
00191
00192   moveForward(-15, 80);
00193
00194   return true;
00195 }
```

2.1.2.7 badAuton()

```
void badAuton (
              Side side,
              Color color )
```

Definition at line 198 of file auton.cpp.

```
00198                                     {
00199
00200   if(color == Color::BLUE && side == Side::LEFT)
00201     badAutonBlueLeft();
00202   else if (color == Color::BLUE && side == Side::RIGHT)
00203     badAutonBlueRight();
00204   else if (color == Color::RED && side == Side::LEFT)
00205     badAutonRedLeft();
00206   else if (color == Color::RED && side == Side::RIGHT)
00207     badAutonRedRight();
00208 }
```

2.1.2.8 badAutonBlueLeft()

```
void badAutonBlueLeft ( )
```

Definition at line 211 of file auton.cpp.

```
00211                       {
00212
00213 }
```

2.1.2.9 badAutonBlueRight()

```
void badAutonBlueRight ( )
```

Definition at line 215 of file auton.cpp.

```
00215                        {
00216
00217 }
```

2.1.2.10 badAutonRedLeft()

```
void badAutonRedLeft ( )
```

Definition at line 219 of file auton.cpp.

```
00219                       {
00220
00221 }
```

2.1.2.11 badAutonRedRight()

```
void badAutonRedRight ( )
```

Definition at line 223 of file auton.cpp.

```
00223                        {
00224
00225 }
```

2.1.2.12 moveForward()

```
bool moveForward (
            double inches,
            double speed,
            bool blocking )
```

Declares the autonomous functions for the robot

Author

Michael Baraty

Date

11/9/2019 Moves the robot forward for a certain distance

Parameters

inches	The distance to move in inches (negative for reverse)

Author

Michael Baraty

Date

11/9/2019

Definition at line 5 of file auton.cpp.

```
00005                                                                    {
00006
00007
00008   double rotations = inches * ROTATIONS_PER_INCH;
00009   if(blocking) {
00010     MOTOR_BACK_LEFT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev, speed,
       velocityUnits::pct);
00011     MOTOR_BACK_RIGHT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev,
      speed, velocityUnits::pct);
00012     MOTOR_FRONT_LEFT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev,
      speed, velocityUnits::pct);
00013     MOTOR_FRONT_RIGHT.rotateFor(directionType::fwd, rotations, rotationUnits::rev, speed,
      velocityUnits::pct);
00014   } else {
00015     MOTOR_BACK_LEFT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev, speed,
       velocityUnits::pct);
00016     MOTOR_BACK_RIGHT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev,
      speed, velocityUnits::pct);
00017     MOTOR_FRONT_LEFT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev,
      speed, velocityUnits::pct);
00018     MOTOR_FRONT_RIGHT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev,
      speed, velocityUnits::pct);
00019   }
00020   return true;
00021 }
```

2.1.2.13 pivotClockwise()

```
bool pivotClockwise (
              float degrees,
              bool blocking )
```

Pivots the robot clockwise to a certain angle

Parameters

degrees	The number of degrees to pivot the robot

Author

Michael Baraty

Date

11/9/2019

Definition at line 23 of file auton.cpp.

```
00023                                                          {
00024   double rotations_per_360 = 6.4;
00025   double rotations = rotations_per_360 * (degrees / 360);
00026
00027   if(blocking){
00028   MOTOR_BACK_LEFT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00029   MOTOR_BACK_RIGHT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00030   MOTOR_FRONT_LEFT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00031   MOTOR_FRONT_RIGHT.rotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00032   } else {
00033     MOTOR_BACK_LEFT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00034     MOTOR_BACK_RIGHT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct)
    ;
00035     MOTOR_FRONT_LEFT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00036     MOTOR_FRONT_RIGHT.startRotateFor(-rotations, rotationUnits::rev, 80,
     velocityUnits::pct);
00037   }
00038
00039   return true;
00040 }
```

2.1.2.14 pivotCounterClockwise()

```
bool pivotCounterClockwise (
             float degrees,
             bool blocking )
```

Pivots the robot counter-clockwise to a certain angle

Parameters

degrees	The number of degrees to pivot the robot

Author

Michael Baraty

Date

11/9/2019

Definition at line 42 of file auton.cpp.

```
00042                                                                {
00043   double rotations_per_360 = 6.4;
00044   double rotations = rotations_per_360 * (degrees / 360);
00045
00046   if(blocking) {
00047   MOTOR_BACK_LEFT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00048   MOTOR_BACK_RIGHT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00049   MOTOR_FRONT_LEFT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00050   MOTOR_FRONT_RIGHT.rotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00051   } else {
00052     MOTOR_BACK_LEFT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00053   MOTOR_BACK_RIGHT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00054   MOTOR_FRONT_LEFT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00055   MOTOR_FRONT_RIGHT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00056   }
00057   return true;
00058 }
```

2.2 auton.h

```
00001
00006 #ifndef AUTON_H
00007 #define AUTON_H
00008
00015 bool moveForward(double inches, double speed, bool blocking);
00016
00023 bool pivotClockwise(float degrees, bool blocking);
00024
00031 bool pivotCounterClockwise(float degrees, bool blocking);
00032
00038 enum class Color : bool {
00039   BLUE = false,
00040   RED = true
00041 };
00042
00048 enum class Side : bool {
00049   LEFT = false,
00050   RIGHT = true
00051 };
00052
00060 void auton(Side side, Color color);
00061
00067 bool autonStart();
00068
00074 void autonBlueLeft();
00075
00081 void autonBlueRight();
00082
00088 void autonRedLeft();
00089
00095 void autonRedRight();
00096
00097 void badAuton(Side side, Color color);
00098
00099 void badAutonBlueLeft();
00100
00101 void badAutonBlueRight();
00102
00103 void badAutonRedLeft();
00104
00105 void badAutonRedRight();
00106
00107 #endif
```

2.3 include/controller.h File Reference

```
#include "vex.h"
```

Functions

- int axisValue (controller::axis Axis)
- bool buttonIsPressed (controller::button Button)

2.3.1 Function Documentation

2.3.1.1 axisValue()

```
int axisValue (
            controller::axis Axis )
```

Returns an axis value of the controller in a -100 - 100 scale, designed to be used with motor speeds in percent

Parameters

Axis	the axis on the controller that will be read (Ex. [controller].axis3 for the left x axis)

Author

Michael Baraty

Date

11/9/2019

Definition at line 4 of file controller.cpp.

```
00004                                   {
00005   return Axis.position();
00006 }
```

2.3.1.2 buttonIsPressed()

```
bool buttonIsPressed (
            controller::button Button )
```

Returns a boolean for whether a designated button is being pressed

Parameters

Button	the button on the controller that will be read (Ex. [controller].buttonA for the A button)

Author

Michael Baraty

Date

11/9/2019

Definition at line 8 of file controller.cpp.

```
00008                                                {
00009   return Button.pressing();
00010 }
```

2.4 controller.h

```
00001
00006 #ifndef CONTROLLER_H
00007 #define CONTROLLER_H
00008
00009 #include "vex.h"
00010
00011 using namespace vex;
00012
00019 int axisValue (controller::axis Axis);
00020
00027 bool buttonIsPressed (controller::button Button);
00028
00029 #endif
```

2.5 include/declarations.h File Reference

Variables

- controller MASTER
- motor MOTOR_BACK_LEFT
- motor MOTOR_BACK_RIGHT
- motor MOTOR_FRONT_LEFT
- motor MOTOR_FRONT_RIGHT
- motor MOTOR_INTAKE_A
- motor MOTOR_INTAKE_B
- motor MOTOR_STACK
- motor MOTOR_ARM
- static int THRESHOLD = 5
- static float SPEED_MULTIPLIER = .9
- static float WHEEL_DIAMETER = 4
- static double INCHES_PER_ROTATION = 12.56
- static float ROTATIONS_PER_INCH = .07962

2.5.1 Variable Documentation

2.5.1.1 INCHES_PER_ROTATION

```
double INCHES_PER_ROTATION = 12.56  [static]
```

Declares the inches the robot will move per single rotation of the wheel to facilitate autonomous

Author

Michael Baraty

Date

11/9/2019

Definition at line 99 of file declarations.h.

2.5.1.2 MASTER

```
controller MASTER
```

Makes the main controller accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 18 of file main.cpp.

2.5.1.3 MOTOR_ARM

```
motor MOTOR_ARM
```

Makes the intake lifter motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 29 of file main.cpp.

2.5.1.4 MOTOR_BACK_LEFT

`motor MOTOR_BACK_LEFT`

Makes the back left motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 22 of file main.cpp.

2.5.1.5 MOTOR_BACK_RIGHT

`motor MOTOR_BACK_RIGHT`

Makes the back right motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 23 of file main.cpp.

2.5.1.6 MOTOR_FRONT_LEFT

`motor MOTOR_FRONT_LEFT`

Makes the front left motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 24 of file main.cpp.

2.5.1.7 MOTOR_FRONT_RIGHT

```
motor MOTOR_FRONT_RIGHT
```

Makes the front right motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 25 of file main.cpp.

2.5.1.8 MOTOR_INTAKE_A

```
motor MOTOR_INTAKE_A
```

Makes the right intake motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 26 of file main.cpp.

2.5.1.9 MOTOR_INTAKE_B

```
motor MOTOR_INTAKE_B
```

Makes the left intake motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 27 of file main.cpp.

2.5.1.10 MOTOR_STACK

```
motor MOTOR_STACK
```

Makes the stack mechanism motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 28 of file main.cpp.

2.5.1.11 ROTATIONS_PER_INCH

```
float ROTATIONS_PER_INCH = .07962  [static]
```

Declares the number of rotations that it will take to move the robot an inch

Author

Michael Baraty

Date

11/9/2019

Definition at line 106 of file declarations.h.

2.5.1.12 SPEED_MULTIPLIER

```
float SPEED_MULTIPLIER = .9  [static]
```

Declares the speed multiplier for the drivebase of the robot

Author

Michael Baraty

Date

11/9/2019

Definition at line 85 of file declarations.h.

2.5.1.13 THRESHOLD

```
int THRESHOLD = 5  [static]
```

Declares the minimum motor power to move that cancels out friction

Author

Michael Baraty

Date

11/9/2019

Definition at line 78 of file declarations.h.

2.5.1.14 WHEEL_DIAMETER

```
float WHEEL_DIAMETER = 4  [static]
```

Declares the diameter of each wheel to facilitate autonomous programming

Author

Michael Baraty

Date

11/9/2019

Definition at line 92 of file declarations.h.

2.6 declarations.h

```
00001
00006 #ifndef DECLARATIONS_H
00007 #define DECLARATIONS_H
00008 using namespace vex;
00009
00015 extern controller MASTER;
00016
00022 extern motor MOTOR_BACK_LEFT;
00023
00029 extern motor MOTOR_BACK_RIGHT;
00030
00036 extern  motor MOTOR_FRONT_LEFT;
00037
00043 extern motor MOTOR_FRONT_RIGHT;
00044
00050 extern motor MOTOR_INTAKE_A;
00051
00057 extern motor MOTOR_INTAKE_B;
00058
00064 extern motor MOTOR_STACK;
00065
00071 extern motor MOTOR_ARM;
00072
00078 static int THRESHOLD = 5;
00079
00085 static float SPEED_MULTIPLIER = .9;
00086
00092 static float WHEEL_DIAMETER = 4;
00093
00099 static double INCHES_PER_ROTATION = 12.56;
00100
00106 static float ROTATIONS_PER_INCH = .07962;
00107
00108
00109 #endif
```

2.7 include/drive.h File Reference

```
#include "controller.h"
```

Enumerations

- enum DriveSide : int { DriveSide::LEFT = 0, DriveSide::RIGHT = 1, DriveSide::BOTH = 2 }

Functions

- void arcadeDrive ()
- void tankDrive ()
- void setSideSpeed (DriveSide side, int speed)
- void drive ()
- void moveStackForward ()
- void moveStackBack ()
- void stackControl ()
- void armUp ()
- void armDown ()
- void armControl ()
- void intakeIn ()
- void intakeOut ()
- void intakeControl ()

2.7.1 Enumeration Type Documentation

2.7.1.1 DriveSide

```
enum DriveSide : int [strong]
```

Declares all the basic movement functions for the robot

Author

Michael Baraty

Date

11/9/2019 Declares the different possible motor combinations to drive the robot

Author

Michael Baraty

Date

11/9/2019

Enumerator

LEFT	
RIGHT	
BOTH	

Definition at line 16 of file drive.h.

```
00016                       : int {
00017     LEFT = 0,
00018     RIGHT = 1,
00019     BOTH = 2
00020   };
```

2.7.2 Function Documentation

2.7.2.1 arcadeDrive()

```
void arcadeDrive ( )
```

Initiates the arcade control configuration for the controller, with the left y axis for linear movement and the right x axis for pivoting

Author

Michael Baraty

Date

11/9/2019

Definition at line 37 of file drive.cpp.

```
00037                     {
00038   int x = SPEED_MULTIPLIER  * -axisValue(MASTER.Axis1);
00039   int y = SPEED_MULTIPLIER * (.7  * (pow(-axisValue(
      MASTER.Axis3) / 9, 3) / 10));
00040   int speedLeft = abs(x + y) > THRESHOLD? -(x + y): 0;
00041   int speedRight = abs(x - y) > THRESHOLD? (x - y): 0;
00042
00043   if(slowMode){
00044     setSideSpeed(DriveSide::LEFT, speedLeft / 3);
00045      setSideSpeed(DriveSide::RIGHT, speedRight / 3);
00046   } else {
00047     setSideSpeed(DriveSide::LEFT, speedLeft);
00048     setSideSpeed(DriveSide::RIGHT, speedRight);
00049   }
00050 }
```

2.7.2.2 armControl()

```
void armControl ( )
```

Reads the controller's button inputs to initiate the arm lifter's movement while the button is being pressed

Author

Michael Baraty

Date

11/9/2019

Definition at line 97 of file drive.cpp.

```
00097                 {
00098   if(buttonIsPressed(MASTER.ButtonR1)){
00099     armUp();
00100   }
00101   else if (buttonIsPressed(MASTER.ButtonR2)) {
00102     armDown();
00103   }
00104   else {
00105     MOTOR_ARM.stop(brakeType::hold);
00106   }
00107 }
```

2.7.2.3 armDown()

```
void armDown ( )
```

Moves the intake lifter down to a specified position

Author

Michael Baraty

Date

11/9/2019

Definition at line 92 of file drive.cpp.

```
00092               {
00093   MOTOR_ARM.startSpinTo(-10, rotationUnits::rev, 100, velocityUnits::pct);
00094   //realValue 0
00095 }
```

2.7.2.4 armUp()

```
void armUp ( )
```

Moves the intake lifter up to a specified position

Author

Michael Baraty

Date

11/9/2019

Definition at line 87 of file drive.cpp.

```
00087             {
00088   MOTOR_ARM.startSpinTo(60.6, rotationUnits::rev, 100, velocityUnits::pct);
00089   //realValue 6.6
00090 }
```

2.7.2.5 drive()

```
void drive ( )
```

Initiates the drive code for the entire robot

Author

Michael Baraty

Date

11/9/2019

Definition at line 7 of file drive.cpp.

```
00007             {
00008   arcadeDrive();
00009   stackControl();
00010   armControl();
00011   intakeControl();
00012
00013   if(buttonIsPressed(MASTER.ButtonUp)){
00014     slowMode = false;
00015   } else if(buttonIsPressed(MASTER.ButtonDown)){
00016     slowMode = true;
00017   }
00018
00019 }
```

2.7.2.6 intakeControl()

```
void intakeControl ( )
```

Reads the controller's button inputs to spin the intake while the button is being pressed

Author

Michael Baraty

Date

11/9/2019

Definition at line 129 of file drive.cpp.

```
00129                       {
00130   if(buttonIsPressed(MASTER.ButtonL1)){
00131     intakeIn();
00132   }
00133   else if (buttonIsPressed(MASTER.ButtonL2)) {
00134     intakeOut();
00135   }
00136   else {
00137     MOTOR_INTAKE_A.stop(brakeType::hold);
00138     MOTOR_INTAKE_B.stop(brakeType::hold);
00139   }
00140 }
```

2.7.2.7 intakeIn()

```
void intakeIn ( )
```

Spins the intake to intake the cubes

Author

Michael Baraty

Date

11/9/2019

Definition at line 109 of file drive.cpp.

```
00109                   {
00110   if(!slowMode) {
00111     MOTOR_INTAKE_A.spin(directionType::fwd, 100, velocityUnits::pct);
00112     MOTOR_INTAKE_B.spin(directionType::fwd, 100, velocityUnits::pct);
00113   } else {
00114     MOTOR_INTAKE_A.spin(directionType::fwd, (100), velocityUnits::pct);
00115     MOTOR_INTAKE_B.spin(directionType::fwd, (100), velocityUnits::pct);
00116   }
00117 }
```

2.7.2.8 intakeOut()

```
void intakeOut ( )
```

Spins the intake to eject the cubes

Author

Michael Baraty

Date

11/9/2019

Definition at line 119 of file drive.cpp.

```
00119                 {
00120    if(!slowMode) {
00121     MOTOR_INTAKE_A.spin(directionType::rev, 100, velocityUnits::pct);
00122     MOTOR_INTAKE_B.spin(directionType::rev, 100, velocityUnits::pct);
00123   } else {
00124     MOTOR_INTAKE_A.spin(directionType::rev, .5*(100), velocityUnits::pct);
00125     MOTOR_INTAKE_B.spin(directionType::rev, .5*(100), velocityUnits::pct);
00126   }
00127 }
```

2.7.2.9 moveStackBack()

```
void moveStackBack ( )
```

Moves the stack backwards to a specified position

Author

Michael Baraty

Date

11/9/2019

Definition at line 72 of file drive.cpp.

```
00072                     {
00073   double final = 0;
00074   MOTOR_STACK.startSpinTo(-10, rotationUnits::rev, 80, velocityUnits::pct);
00075 }
```

2.7.2.10 moveStackForward()

```
void moveStackForward ( )
```

Moves the stack forward to a specified position

Author

Michael Baraty

Date

11/9/2019

Definition at line 67 of file drive.cpp.

```
00067                           {
00068   double final = 1.5;
00069   MOTOR_STACK.startSpinTo(10, rotationUnits::rev, 30, velocityUnits::pct);
00070 }
```

2.7.2.11 setSideSpeed()

```
void setSideSpeed (
              DriveSide side,
              int speed )
```

Sets the speed of the designated drive side

Parameters

side	The DriveSide that is going to be powered
speed	The speed that the robot will move at between -100 - 100

Author

Michael Baraty

Date

11/9/2019

Definition at line 21 of file drive.cpp.

```
00021                                           {
00022   if(side == DriveSide::LEFT) {
00023     MOTOR_BACK_LEFT.spin(directionType::fwd, speed, velocityUnits::pct);
00024     MOTOR_FRONT_LEFT.spin(directionType::fwd, speed, velocityUnits::pct);
00025   } else if (side == DriveSide::RIGHT) {
00026     MOTOR_BACK_RIGHT.spin(directionType::fwd, speed, velocityUnits::pct);
00027     MOTOR_FRONT_RIGHT.spin(directionType::fwd, speed, velocityUnits::pct);
00028   }
00029   else {
00030     MOTOR_FRONT_LEFT.spin(directionType::fwd, speed, velocityUnits::pct);
00031     MOTOR_FRONT_RIGHT.spin(directionType::fwd, speed, velocityUnits::pct);
00032     MOTOR_BACK_RIGHT.spin(directionType::fwd, speed, velocityUnits::pct);
00033     MOTOR_BACK_LEFT.spin(directionType::fwd, speed, velocityUnits::pct);
00034   }
00035 }
```

2.7.2.12 stackControl()

```
void stackControl ( )
```

Reads the controller's button inputs to initiate the stack mechanism's movement while the button is being pressed

Author

Michael Baraty

Date

11/9/2019

Definition at line 77 of file drive.cpp.

```
00077                     {
00078   if(buttonIsPressed(MASTER.ButtonX))
00079     moveStackForward();
00080   else if(buttonIsPressed(MASTER.ButtonA))
00081     moveStackBack();
00082   else{
00083     MOTOR_STACK.stop(brakeType::brake);
00084   }
00085 }
```

2.7.2.13 tankDrive()

```
void tankDrive ( )
```

Initiates the tank drive control configuration for the controller, with the left y axis for the left side and the right y axis for the right side

Author

Michael Baraty

Date

11/9/2019

Definition at line 52 of file drive.cpp.

```
00052                 {
00053   int l = SPEED_MULTIPLIER * (.7  * (pow(axisValue(
      MASTER.Axis3) / 9, 3) / 10));
00054   int r = SPEED_MULTIPLIER * (.7  * (pow(axisValue(
      MASTER.Axis2) / 9, 3) / 10));
00055   int speedLeft = abs(l) > THRESHOLD? l: 0;
00056   int speedRight = abs(r) > THRESHOLD? r: 0;
00057
00058   if(slowMode){
00059     setSideSpeed(DriveSide::LEFT, speedLeft / 3);
00060      setSideSpeed(DriveSide::RIGHT, speedRight / 3);
00061   } else {
00062     setSideSpeed(DriveSide::LEFT, speedLeft);
00063     setSideSpeed(DriveSide::RIGHT, speedRight);
00064   }
00065 }
```

2.8 drive.h

```
00001
00006 #ifndef DRIVE_H
00007 #define DRIVE_H
00008
00009 #include "controller.h"
00010
00016 enum class DriveSide : int {
00017     LEFT = 0,
00018     RIGHT = 1,
00019     BOTH = 2
00020  };
00021
00027 void arcadeDrive();
00028
00034 void tankDrive();
00035
00043 void setSideSpeed(DriveSide side, int speed);
00044
00050 void drive();
00051
00057 void moveStackForward();
00058
00064 void moveStackBack();
00065
00071 void stackControl();
00072
00078 void armUp();
00079
00085 void armDown();
00086
00092 void armControl();
00093
00099 void intakeIn();
00100
00106 void intakeOut();
00107
00113 void intakeControl();
00114
00115
00116 #endif
```

2.9 include/init.h File Reference

2.10 init.h

```
00001
00006 #ifndef INIT_H
00007 #define INIT_H
00008
00009
00010 #endif
```

2.11 include/vex.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "v5.h"
#include "v5_vcs.h"
```

2.12 vex.h

```
00001 /*----------------------------------------------------------------------------*/
00002 /*                                                                            */
00003 /*    Module:       vex.h                                                     */
00004 /*    Author:       Vex Robotics                                              */
00005 /*    Created:      1 Feb 2019                                                */
00006 /*    Description:  Default header for V5 projects                            */
00007 /*                                                                            */
00008 /*----------------------------------------------------------------------------*/
00009 //
00010 #ifndef VEX_H
00011 #define VEX_H
00012
00013
00014 #include <stdio.h>
00015 #include <stdlib.h>
00016 #include <string.h>
00017 #include <math.h>
00018
00019 #include "v5.h"
00020 #include "v5_vcs.h"
00021
00022 #endif
```

2.13 src/auton.cpp File Reference

```
#include "auton.h"
#include "drive.h"
#include "declarations.h"
```

Functions

- bool moveForward (double inches, double speed=50, bool blocking=true)
- bool pivotClockwise (float degrees, bool blocking=true)
- bool pivotCounterClockwise (float degrees, bool blocking=true)
- void auton (Side side, Color color)

  the autonomous switcher
- void autonBlueLeft ()
- void autonBlueRight ()
- void autonRedLeft ()
- void autonRedRight ()
- bool autonStart ()
- void badAuton (Side side, Color color)
- void badAutonBlueLeft ()
- void badAutonBlueRight ()
- void badAutonRedLeft ()
- void badAutonRedRight ()

2.13.1 Function Documentation

2.13.1.1 auton()

```
void auton (
            Side side,
            Color color )
```

the autonomous switcher

Author

Michael Baraty

Definition at line 64 of file auton.cpp.

```
00064                                    {
00065   autonStart();
00066
00067   if(color == Color::BLUE && side == Side::LEFT)
00068     autonBlueLeft();
00069   else if (color == Color::BLUE && side == Side::RIGHT)
00070     autonBlueRight();
00071   else if (color == Color::RED && side == Side::LEFT)
00072     autonRedLeft();
00073   else if (color == Color::RED && side == Side::RIGHT)
00074     autonRedRight();
00075 }
```

2.13.1.2 autonBlueLeft()

```
void autonBlueLeft ( )
```

Initiates blue left autonomous routine

Author

Michael Baraty

Date

11/9/2019

Definition at line 78 of file auton.cpp.

```
00078                   {
00079   MOTOR_INTAKE_A.startRotateFor(7, rotationUnits::rev, 100, velocityUnits::pct);
00080   MOTOR_INTAKE_B.startRotateFor(7, rotationUnits::rev, 100, velocityUnits::pct);
00081   moveForward(35, 33);
00082   /*pivotClockwise(190);
00083   moveForward(24);
00084   pivotCounterClockwise(45);*/
00085   moveForward(-24, 60);
00086   pivotCounterClockwise(135);
00087   moveForward(10);
00088   moveForward(5, 30, false);
00089   MOTOR_INTAKE_A.startRotateFor(-.5, rotationUnits::rev, 100, velocityUnits::pct);
00090   MOTOR_INTAKE_B.startRotateFor(-.5, rotationUnits::rev, 100, velocityUnits::pct);
00091   MOTOR_STACK.rotateFor(2, rev);
00092   MOTOR_INTAKE_A.startRotateFor(-20, rotationUnits::rev, 100, velocityUnits::pct);
00093   MOTOR_INTAKE_B.startRotateFor(-20, rotationUnits::rev, 100, velocityUnits::pct);
00094   moveForward(-20, 33);
00095   MOTOR_INTAKE_A.stop();
00096   MOTOR_INTAKE_B.stop();
00097 }
```

2.13.1.3 autonBlueRight()

```
void autonBlueRight ( )
```

Initiates blue right autonomous routine

Author

Michael Baraty

Date

11/9/2019

Definition at line 99 of file auton.cpp.

```
00099                     {
00100   intakeIn();
00101   moveForward(25, 70);
00102   moveForward(12, 40);
00103   vexDelay(1000);
00104   MOTOR_INTAKE_A.stop(hold);
00105   MOTOR_INTAKE_B.stop(hold);
00106   moveForward(-28, 80);
00107   pivotClockwise(130);
00108   moveForward(29, 50);
00109   //MOTOR_INTAKE_A.rotateFor(-.25, rotationUnits::rev, 100, velocityUnits::pct);
00110   //MOTOR_INTAKE_B.rotateFor(-.25, rotationUnits::rev, 100, velocityUnits::pct);
00111   //MOTOR_STACK.startRotateTo(2.9, rotationUnits::rev);
00112   //vexDelay(1500);
00113
00114   MOTOR_INTAKE_A.startRotateFor(-10, rotationUnits::rev, 100, velocityUnits::pct);
00115   MOTOR_INTAKE_B.startRotateFor(-10, rotationUnits::rev, 100, velocityUnits::pct);
00116   pivotClockwise(60, true);
00117
00118   moveForward(-13, 33, true);
00119
00120   MOTOR_STACK.rotateTo(0, rotationUnits::rev, 80, velocityUnits::pct);
00121 }
```

2.13.1.4 autonRedLeft()

```
void autonRedLeft ( )
```

Initiates red left autonomous routine

Author

Michael Baraty

Date

11/9/2019

Definition at line 123 of file auton.cpp.

```
00123                     {
00124   intakeIn();
00125   moveForward(25, 70);
00126   moveForward(12, 40);
00127   vexDelay(1000);
00128   MOTOR_INTAKE_A.stop(hold);
00129   MOTOR_INTAKE_B.stop(hold);
00130   moveForward(-28, 80);
00131   pivotCounterClockwise(130);
00132   moveForward(29, 50);
00133   //MOTOR_INTAKE_A.rotateFor(-.25, rotationUnits::rev, 100, velocityUnits::pct);
00134   //MOTOR_INTAKE_B.rotateFor(-.25, rotationUnits::rev, 100, velocityUnits::pct);
00135   //MOTOR_STACK.startRotateTo(2.9, rotationUnits::rev);
00136   //vexDelay(1500);
00137
00138   MOTOR_INTAKE_A.startRotateFor(-10, rotationUnits::rev, 100, velocityUnits::pct);
00139   MOTOR_INTAKE_B.startRotateFor(-10, rotationUnits::rev, 100, velocityUnits::pct);
00140   pivotCounterClockwise(60, true);
00141
00142   moveForward(-13, 33, true);
00143
00144   MOTOR_STACK.rotateTo(0, rotationUnits::rev, 80, velocityUnits::pct);
00145 }
```

2.13.1.5 autonRedRight()

```
void autonRedRight ( )
```

Initiates red right autonomous routine

Author

Michael Baraty

Date

11/9/2019

Definition at line 147 of file auton.cpp.

```
00147                   {
00148   MOTOR_INTAKE_A.startRotateFor(7, rotationUnits::rev, 100, velocityUnits::pct);
00149   MOTOR_INTAKE_B.startRotateFor(7, rotationUnits::rev, 100, velocityUnits::pct);
00150   moveForward(35, 33);
00151   /*pivotClockwise(190);
00152   moveForward(24);
00153   pivotCounterClockwise(45);*/
00154   moveForward(-24, 60);
00155   pivotClockwise(135);
00156   moveForward(10);
00157   moveForward(5, 30, false);
00158   MOTOR_INTAKE_A.startRotateFor(-.5, rotationUnits::rev, 100, velocityUnits::pct);
00159   MOTOR_INTAKE_B.startRotateFor(-.5, rotationUnits::rev, 100, velocityUnits::pct);
00160   MOTOR_STACK.rotateFor(2, rev);
00161   MOTOR_INTAKE_A.startRotateFor(-20, rotationUnits::rev, 100, velocityUnits::pct);
00162   MOTOR_INTAKE_B.startRotateFor(-20, rotationUnits::rev, 100, velocityUnits::pct);
00163   moveForward(-20, 33);
00164   MOTOR_INTAKE_A.stop();
00165   MOTOR_INTAKE_B.stop();
00166 }
```

2.13.1.6 autonStart()

```
bool autonStart ( )
```

1. Move intake up so mechanisms deploy
2. Move intake back down
3. Run intake and drive forward to pick up preload
4. Score cubes

Definition at line 174 of file auton.cpp.

```
00174                   {
00175
00176   MOTOR_INTAKE_A.startRotateFor(directionType::rev, 3, rotationUnits::rev, 100,
      velocityUnits::pct);
00177   MOTOR_INTAKE_B.startRotateFor(directionType::rev, 3, rotationUnits::rev, 100,
      velocityUnits::pct);
00178   MOTOR_ARM.rotateTo(2, rotationUnits::rev, 100, velocityUnits::pct);
00179
00180   MOTOR_STACK.startRotateTo(1.5, rev);
00181
00182   MOTOR_ARM.rotateTo(3.9, rotationUnits::rev, 100, velocityUnits::pct);
00183
00184
00185   moveForward(3);
00186
00187   MOTOR_STACK.startRotateTo(0, rev);
00188
00189   MOTOR_ARM.rotateTo(-.15, rotationUnits::rev, 100, velocityUnits::pct);
00190   MOTOR_ARM.stop(hold);
00191
00192   moveForward(-15, 80);
00193
00194   return true;
00195 }
```

2.13.1.7 badAuton()

```
void badAuton (
              Side side,
              Color color )
```

Definition at line 198 of file auton.cpp.

```
00198                                     {
00199
00200   if(color == Color::BLUE && side == Side::LEFT)
00201     badAutonBlueLeft();
00202   else if (color == Color::BLUE && side == Side::RIGHT)
00203     badAutonBlueRight();
00204   else if (color == Color::RED && side == Side::LEFT)
00205     badAutonRedLeft();
00206   else if (color == Color::RED && side == Side::RIGHT)
00207     badAutonRedRight();
00208 }
```

2.13.1.8 badAutonBlueLeft()

```
void badAutonBlueLeft ( )
```

Definition at line 211 of file auton.cpp.

```
00211                       {
00212
00213 }
```

2.13.1.9 badAutonBlueRight()

```
void badAutonBlueRight ( )
```

Definition at line 215 of file auton.cpp.

```
00215                         {
00216
00217 }
```

2.13.1.10 badAutonRedLeft()

```
void badAutonRedLeft ( )
```

Definition at line 219 of file auton.cpp.

```
00219                        {
00220
00221 }
```

2.13.1.11 badAutonRedRight()

```
void badAutonRedRight ( )
```

Definition at line 223 of file auton.cpp.

```
00223                         {
00224
00225 }
```

2.13.1.12 moveForward()

```
bool moveForward (
            double inches,
            double speed,
            bool blocking )
```

Declares the autonomous functions for the robot

Author

Michael Baraty

Date

11/9/2019 Moves the robot forward for a certain distance

Parameters

inches	The distance to move in inches (negative for reverse)

Author

Michael Baraty

Date

11/9/2019

Definition at line 5 of file auton.cpp.

```
00005                                                                    {
00006
00007
00008   double rotations = inches * ROTATIONS_PER_INCH;
00009   if(blocking) {
00010     MOTOR_BACK_LEFT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev, speed,
       velocityUnits::pct);
00011     MOTOR_BACK_RIGHT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev,
      speed, velocityUnits::pct);
00012     MOTOR_FRONT_LEFT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev,
      speed, velocityUnits::pct);
00013     MOTOR_FRONT_RIGHT.rotateFor(directionType::fwd, rotations, rotationUnits::rev, speed,
      velocityUnits::pct);
00014   } else {
00015     MOTOR_BACK_LEFT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev, speed,
       velocityUnits::pct);
00016     MOTOR_BACK_RIGHT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev,
      speed, velocityUnits::pct);
00017     MOTOR_FRONT_LEFT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev,
      speed, velocityUnits::pct);
00018     MOTOR_FRONT_RIGHT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev,
      speed, velocityUnits::pct);
00019   }
00020   return true;
00021 }
```

2.13.1.13 pivotClockwise()

```
bool pivotClockwise (
              float degrees,
              bool blocking )
```

Pivots the robot clockwise to a certain angle

Parameters

degrees	The number of degrees to pivot the robot

Author

Michael Baraty

Date

11/9/2019

Definition at line 23 of file auton.cpp.

```
00023                                                          {
00024   double rotations_per_360 = 6.4;
00025   double rotations = rotations_per_360 * (degrees / 360);
00026
00027   if(blocking){
00028   MOTOR_BACK_LEFT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00029   MOTOR_BACK_RIGHT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00030   MOTOR_FRONT_LEFT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00031   MOTOR_FRONT_RIGHT.rotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00032   } else {
00033     MOTOR_BACK_LEFT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00034     MOTOR_BACK_RIGHT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct)
    ;
00035     MOTOR_FRONT_LEFT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00036     MOTOR_FRONT_RIGHT.startRotateFor(-rotations, rotationUnits::rev, 80,
     velocityUnits::pct);
00037   }
00038
00039   return true;
00040 }
```

2.13.1.14 pivotCounterClockwise()

```
bool pivotCounterClockwise (
             float degrees,
             bool blocking )
```

Pivots the robot counter-clockwise to a certain angle

Parameters

degrees	The number of degrees to pivot the robot

Author

Michael Baraty

Date

11/9/2019

Definition at line 42 of file auton.cpp.

```
00042                                                                    {
00043   double rotations_per_360 = 6.4;
00044   double rotations = rotations_per_360 * (degrees / 360);
00045
00046   if(blocking) {
00047   MOTOR_BACK_LEFT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00048   MOTOR_BACK_RIGHT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00049   MOTOR_FRONT_LEFT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00050   MOTOR_FRONT_RIGHT.rotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00051   } else {
00052     MOTOR_BACK_LEFT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00053   MOTOR_BACK_RIGHT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00054   MOTOR_FRONT_LEFT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00055   MOTOR_FRONT_RIGHT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00056   }
00057   return true;
00058 }
```

2.14 auton.cpp

```
00001 #include "auton.h"
00002 #include "drive.h"
00003 #include "declarations.h"
00004
00005 bool moveForward(double inches, double speed = 50, bool blocking = true) {
00006
00007
00008   double rotations = inches * ROTATIONS_PER_INCH;
00009   if(blocking) {
00010     MOTOR_BACK_LEFT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev, speed,
       velocityUnits::pct);
00011     MOTOR_BACK_RIGHT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev,
      speed, velocityUnits::pct);
00012     MOTOR_FRONT_LEFT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev,
      speed, velocityUnits::pct);
00013     MOTOR_FRONT_RIGHT.rotateFor(directionType::fwd, rotations, rotationUnits::rev, speed,
      velocityUnits::pct);
00014   } else {
00015     MOTOR_BACK_LEFT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev, speed,
       velocityUnits::pct);
00016     MOTOR_BACK_RIGHT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev,
      speed, velocityUnits::pct);
00017     MOTOR_FRONT_LEFT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev,
      speed, velocityUnits::pct);
00018     MOTOR_FRONT_RIGHT.startRotateFor(directionType::fwd, rotations, rotationUnits::rev,
      speed, velocityUnits::pct);
00019   }
00020   return true;
00021 }
00022
00023 bool pivotClockwise(float degrees, bool blocking = true) {
00024   double rotations_per_360 = 6.4;
00025   double rotations = rotations_per_360 * (degrees / 360);
00026
00027   if(blocking){
00028   MOTOR_BACK_LEFT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00029   MOTOR_BACK_RIGHT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00030   MOTOR_FRONT_LEFT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00031   MOTOR_FRONT_RIGHT.rotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00032   } else {
00033     MOTOR_BACK_LEFT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00034     MOTOR_BACK_RIGHT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct)
      ;
00035     MOTOR_FRONT_LEFT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00036     MOTOR_FRONT_RIGHT.startRotateFor(-rotations, rotationUnits::rev, 80,
      velocityUnits::pct);
00037   }
00038
00039   return true;
00040 }
00041
00042 bool pivotCounterClockwise(float degrees, bool blocking = true) {
00043   double rotations_per_360 = 6.4;
00044   double rotations = rotations_per_360 * (degrees / 360);
00045
00046   if(blocking) {
00047   MOTOR_BACK_LEFT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
```

```
00048   MOTOR_BACK_RIGHT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00049   MOTOR_FRONT_LEFT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00050   MOTOR_FRONT_RIGHT.rotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00051   } else {
00052     MOTOR_BACK_LEFT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00053   MOTOR_BACK_RIGHT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00054   MOTOR_FRONT_LEFT.startRotateFor(-rotations, rotationUnits::rev, 80, velocityUnits::pct);
00055   MOTOR_FRONT_RIGHT.startRotateFor(rotations, rotationUnits::rev, 80, velocityUnits::pct);
00056   }
00057   return true;
00058 }
00059
00064 void auton(Side side, Color color) {
00065   autonStart();
00066
00067   if(color == Color::BLUE && side == Side::LEFT)
00068     autonBlueLeft();
00069   else if (color == Color::BLUE && side == Side::RIGHT)
00070     autonBlueRight();
00071   else if (color == Color::RED && side == Side::LEFT)
00072     autonRedLeft();
00073   else if (color == Color::RED && side == Side::RIGHT)
00074     autonRedRight();
00075 }
00076
00077
00078 void autonBlueLeft(){
00079   MOTOR_INTAKE_A.startRotateFor(7, rotationUnits::rev, 100, velocityUnits::pct);
00080   MOTOR_INTAKE_B.startRotateFor(7, rotationUnits::rev, 100, velocityUnits::pct);
00081   moveForward(35, 33);
00082   /*pivotClockwise(190);
00083   moveForward(24);
00084   pivotCounterClockwise(45);*/
00085   moveForward(-24, 60);
00086   pivotCounterClockwise(135);
00087   moveForward(10);
00088   moveForward(5, 30, false);
00089   MOTOR_INTAKE_A.startRotateFor(-.5, rotationUnits::rev, 100, velocityUnits::pct);
00090   MOTOR_INTAKE_B.startRotateFor(-.5, rotationUnits::rev, 100, velocityUnits::pct);
00091   MOTOR_STACK.rotateFor(2, rev);
00092   MOTOR_INTAKE_A.startRotateFor(-20, rotationUnits::rev, 100, velocityUnits::pct);
00093   MOTOR_INTAKE_B.startRotateFor(-20, rotationUnits::rev, 100, velocityUnits::pct);
00094   moveForward(-20, 33);
00095   MOTOR_INTAKE_A.stop();
00096   MOTOR_INTAKE_B.stop();
00097 }
00098
00099 void autonBlueRight(){
00100   intakeIn();
00101   moveForward(25, 70);
00102   moveForward(12, 40);
00103   vexDelay(1000);
00104   MOTOR_INTAKE_A.stop(hold);
00105   MOTOR_INTAKE_B.stop(hold);
00106   moveForward(-28, 80);
00107   pivotClockwise(130);
00108   moveForward(29, 50);
00109   //MOTOR_INTAKE_A.rotateFor(-.25, rotationUnits::rev, 100, velocityUnits::pct);
00110   //MOTOR_INTAKE_B.rotateFor(-.25, rotationUnits::rev, 100, velocityUnits::pct);
00111   //MOTOR_STACK.startRotateTo(2.9, rotationUnits::rev);
00112   //vexDelay(1500);
00113
00114   MOTOR_INTAKE_A.startRotateFor(-10, rotationUnits::rev, 100, velocityUnits::pct);
00115   MOTOR_INTAKE_B.startRotateFor(-10, rotationUnits::rev, 100, velocityUnits::pct);
00116   pivotClockwise(60, true);
00117
00118   moveForward(-13, 33, true);
00119
00120   MOTOR_STACK.rotateTo(0, rotationUnits::rev, 80, velocityUnits::pct);
00121 }
00122
00123 void autonRedLeft(){
00124   intakeIn();
00125   moveForward(25, 70);
00126   moveForward(12, 40);
00127   vexDelay(1000);
00128   MOTOR_INTAKE_A.stop(hold);
00129   MOTOR_INTAKE_B.stop(hold);
00130   moveForward(-28, 80);
00131   pivotCounterClockwise(130);
00132   moveForward(29, 50);
```

```
00133   //MOTOR_INTAKE_A.rotateFor(-.25, rotationUnits::rev, 100, velocityUnits::pct);
00134   //MOTOR_INTAKE_B.rotateFor(-.25, rotationUnits::rev, 100, velocityUnits::pct);
00135   //MOTOR_STACK.startRotateTo(2.9, rotationUnits::rev);
00136   //vexDelay(1500);
00137
00138   MOTOR_INTAKE_A.startRotateFor(-10, rotationUnits::rev, 100, velocityUnits::pct);
00139   MOTOR_INTAKE_B.startRotateFor(-10, rotationUnits::rev, 100, velocityUnits::pct);
00140   pivotCounterClockwise(60, true);
00141
00142   moveForward(-13, 33, true);
00143
00144   MOTOR_STACK.rotateTo(0, rotationUnits::rev, 80, velocityUnits::pct);
00145 }
00146
00147 void autonRedRight(){
00148   MOTOR_INTAKE_A.startRotateFor(7, rotationUnits::rev, 100, velocityUnits::pct);
00149   MOTOR_INTAKE_B.startRotateFor(7, rotationUnits::rev, 100, velocityUnits::pct);
00150   moveForward(35, 33);
00151   /*pivotClockwise(190);
00152   moveForward(24);
00153   pivotCounterClockwise(45);*/
00154   moveForward(-24, 60);
00155   pivotClockwise(135);
00156   moveForward(10);
00157   moveForward(5, 30, false);
00158   MOTOR_INTAKE_A.startRotateFor(-.5, rotationUnits::rev, 100, velocityUnits::pct);
00159   MOTOR_INTAKE_B.startRotateFor(-.5, rotationUnits::rev, 100, velocityUnits::pct);
00160   MOTOR_STACK.rotateFor(2, rev);
00161   MOTOR_INTAKE_A.startRotateFor(-20, rotationUnits::rev, 100, velocityUnits::pct);
00162   MOTOR_INTAKE_B.startRotateFor(-20, rotationUnits::rev, 100, velocityUnits::pct);
00163   moveForward(-20, 33);
00164   MOTOR_INTAKE_A.stop();
00165   MOTOR_INTAKE_B.stop();
00166 }
00167
00174 bool autonStart() {
00175
00176   MOTOR_INTAKE_A.startRotateFor(directionType::rev, 3, rotationUnits::rev, 100,
      velocityUnits::pct);
00177   MOTOR_INTAKE_B.startRotateFor(directionType::rev, 3, rotationUnits::rev, 100,
      velocityUnits::pct);
00178   MOTOR_ARM.rotateTo(2, rotationUnits::rev, 100, velocityUnits::pct);
00179
00180   MOTOR_STACK.startRotateTo(1.5, rev);
00181
00182   MOTOR_ARM.rotateTo(3.9, rotationUnits::rev, 100, velocityUnits::pct);
00183
00184
00185   moveForward(3);
00186
00187   MOTOR_STACK.startRotateTo(0, rev);
00188
00189   MOTOR_ARM.rotateTo(-.15, rotationUnits::rev, 100, velocityUnits::pct);
00190   MOTOR_ARM.stop(hold);
00191
00192   moveForward(-15, 80);
00193
00194   return true;
00195 }
00196
00197
00198 void badAuton(Side side, Color color) {
00199
00200   if(color == Color::BLUE && side == Side::LEFT)
00201     badAutonBlueLeft();
00202   else if (color == Color::BLUE && side == Side::RIGHT)
00203     badAutonBlueRight();
00204   else if (color == Color::RED && side == Side::LEFT)
00205     badAutonRedLeft();
00206   else if (color == Color::RED && side == Side::RIGHT)
00207     badAutonRedRight();
00208 }
00209
00210
00211 void badAutonBlueLeft(){
00212
00213 }
00214
00215 void badAutonBlueRight(){
00216
00217 }
```

```
00218
00219 void badAutonRedLeft(){
00220
00221 }
00222
00223 void badAutonRedRight(){
00224
00225 }
```

2.15 src/controller.cpp File Reference

```
#include "controller.h"
```

Functions

- int axisValue (controller::axis Axis)
- bool buttonIsPressed (controller::button Button)

2.15.1 Function Documentation

2.15.1.1 axisValue()

```
int axisValue (
            controller::axis Axis )
```

Returns an axis value of the controller in a -100 - 100 scale, designed to be used with motor speeds in percent

Parameters

Axis	the axis on the controller that will be read (Ex. [controller].axis3 for the left x axis)

Author

Michael Baraty

Date

11/9/2019

Definition at line 4 of file controller.cpp.

```
00004                                   {
00005   return Axis.position();
00006 }
```

2.15.1.2 buttonIsPressed()

```
bool buttonIsPressed (
            controller::button Button )
```

Returns a boolean for whether a designated button is being pressed

Parameters

Button	the button on the controller that will be read (Ex. [controller].buttonA for the A button)

Author

Michael Baraty

Date

11/9/2019

Definition at line 8 of file controller.cpp.

```
00008                                                {
00009   return Button.pressing();
00010 }
```

2.16 controller.cpp

```
00001 #include "controller.h"
00002
00003
00004 int axisValue(controller::axis Axis) {
00005   return Axis.position();
00006 }
00007
00008 bool buttonIsPressed(controller::button Button) {
00009   return Button.pressing();
00010 }
00011
00012
00013
00014
00015
```

2.17 src/drive.cpp File Reference

```
#include "drive.h"
#include "declarations.h"
#include "auton.h"
```

Functions

- void drive ()
- void setSideSpeed (DriveSide side, int speed)
- void arcadeDrive ()
- void tankDrive ()
- void moveStackForward ()
- void moveStackBack ()
- void stackControl ()
- void armUp ()
- void armDown ()
- void armControl ()
- void intakeIn ()
- void intakeOut ()
- void intakeControl ()

Variables

- bool slowMode = false

2.17.1 Function Documentation

2.17.1.1 arcadeDrive()

```
void arcadeDrive ( )
```

Initiates the arcade control configuration for the controller, with the left y axis for linear movement and the right x axis for pivoting

Author

Michael Baraty

Date

11/9/2019

Definition at line 37 of file drive.cpp.

```
00037                    {
00038   int x = SPEED_MULTIPLIER  * -axisValue(MASTER.Axis1);
00039   int y = SPEED_MULTIPLIER * (.7  * (pow(-axisValue(
      MASTER.Axis3) / 9, 3) / 10));
00040   int speedLeft = abs(x + y) > THRESHOLD? -(x + y): 0;
00041   int speedRight = abs(x - y) > THRESHOLD? (x - y): 0;
00042
00043   if(slowMode){
00044     setSideSpeed(DriveSide::LEFT, speedLeft / 3);
00045      setSideSpeed(DriveSide::RIGHT, speedRight / 3);
00046   } else {
00047     setSideSpeed(DriveSide::LEFT, speedLeft);
00048     setSideSpeed(DriveSide::RIGHT, speedRight);
00049   }
00050 }
```

2.17.1.2 armControl()

```
void armControl ( )
```

Reads the controller's button inputs to initiate the arm lifter's movement while the button is being pressed

Author

Michael Baraty

Date

11/9/2019

Definition at line 97 of file drive.cpp.

```
00097                 {
00098   if(buttonIsPressed(MASTER.ButtonR1)){
00099     armUp();
00100   }
00101   else if (buttonIsPressed(MASTER.ButtonR2)) {
00102     armDown();
00103   }
00104   else {
00105     MOTOR_ARM.stop(brakeType::hold);
00106   }
00107 }
```

2.17.1.3 armDown()

```
void armDown ( )
```

Moves the intake lifter down to a specified position

Author

Michael Baraty

Date

11/9/2019

Definition at line 92 of file drive.cpp.

```
00092               {
00093   MOTOR_ARM.startSpinTo(-10, rotationUnits::rev, 100, velocityUnits::pct);
00094   //realValue 0
00095 }
```

2.17.1.4 armUp()

```
void armUp ( )
```

Moves the intake lifter up to a specified position

Author

Michael Baraty

Date

11/9/2019

Definition at line 87 of file drive.cpp.

```
00087             {
00088   MOTOR_ARM.startSpinTo(60.6, rotationUnits::rev, 100, velocityUnits::pct);
00089   //realValue 6.6
00090 }
```

2.17.1.5 drive()

```
void drive ( )
```

Initiates the drive code for the entire robot

Author

Michael Baraty

Date

11/9/2019

Definition at line 7 of file drive.cpp.

```
00007             {
00008   arcadeDrive();
00009   stackControl();
00010   armControl();
00011   intakeControl();
00012
00013   if(buttonIsPressed(MASTER.ButtonUp)){
00014     slowMode = false;
00015   } else if(buttonIsPressed(MASTER.ButtonDown)){
00016     slowMode = true;
00017   }
00018
00019 }
```

2.17.1.6 intakeControl()

```
void intakeControl ( )
```

Reads the controller's button inputs to spin the intake while the button is being pressed

Author

Michael Baraty

Date

11/9/2019

Definition at line 129 of file drive.cpp.

```
00129                       {
00130   if(buttonIsPressed(MASTER.ButtonL1)){
00131     intakeIn();
00132   }
00133   else if (buttonIsPressed(MASTER.ButtonL2)) {
00134     intakeOut();
00135   }
00136   else {
00137     MOTOR_INTAKE_A.stop(brakeType::hold);
00138     MOTOR_INTAKE_B.stop(brakeType::hold);
00139   }
00140 }
```

2.17.1.7 intakeIn()

```
void intakeIn ( )
```

Spins the intake to intake the cubes

Author

Michael Baraty

Date

11/9/2019

Definition at line 109 of file drive.cpp.

```
00109                   {
00110   if(!slowMode) {
00111     MOTOR_INTAKE_A.spin(directionType::fwd, 100, velocityUnits::pct);
00112     MOTOR_INTAKE_B.spin(directionType::fwd, 100, velocityUnits::pct);
00113   } else {
00114     MOTOR_INTAKE_A.spin(directionType::fwd, (100), velocityUnits::pct);
00115     MOTOR_INTAKE_B.spin(directionType::fwd, (100), velocityUnits::pct);
00116   }
00117 }
```

2.17.1.8 intakeOut()

```
void intakeOut ( )
```

Spins the intake to eject the cubes

Author

Michael Baraty

Date

11/9/2019

Definition at line 119 of file drive.cpp.

```
00119                 {
00120    if(!slowMode) {
00121     MOTOR_INTAKE_A.spin(directionType::rev, 100, velocityUnits::pct);
00122     MOTOR_INTAKE_B.spin(directionType::rev, 100, velocityUnits::pct);
00123   } else {
00124     MOTOR_INTAKE_A.spin(directionType::rev, .5*(100), velocityUnits::pct);
00125     MOTOR_INTAKE_B.spin(directionType::rev, .5*(100), velocityUnits::pct);
00126   }
00127 }
```

2.17.1.9 moveStackBack()

```
void moveStackBack ( )
```

Moves the stack backwards to a specified position

Author

Michael Baraty

Date

11/9/2019

Definition at line 72 of file drive.cpp.

```
00072                     {
00073   double final = 0;
00074   MOTOR_STACK.startSpinTo(-10, rotationUnits::rev, 80, velocityUnits::pct);
00075 }
```

2.17.1.10 moveStackForward()

```
void moveStackForward ( )
```

Moves the stack forward to a specified position

Author

Michael Baraty

Date

11/9/2019

Definition at line 67 of file drive.cpp.

```
00067                           {
00068   double final = 1.5;
00069   MOTOR_STACK.startSpinTo(10, rotationUnits::rev, 30, velocityUnits::pct);
00070 }
```

2.17.1.11 setSideSpeed()

```
void setSideSpeed (
              DriveSide side,
              int speed )
```

Sets the speed of the designated drive side

Parameters

side	The DriveSide that is going to be powered
speed	The speed that the robot will move at between -100 - 100

Author

Michael Baraty

Date

11/9/2019

Definition at line 21 of file drive.cpp.

```
00021                                           {
00022   if(side == DriveSide::LEFT) {
00023     MOTOR_BACK_LEFT.spin(directionType::fwd, speed, velocityUnits::pct);
00024     MOTOR_FRONT_LEFT.spin(directionType::fwd, speed, velocityUnits::pct);
00025   } else if (side == DriveSide::RIGHT) {
00026     MOTOR_BACK_RIGHT.spin(directionType::fwd, speed, velocityUnits::pct);
00027     MOTOR_FRONT_RIGHT.spin(directionType::fwd, speed, velocityUnits::pct);
00028   }
00029   else {
00030     MOTOR_FRONT_LEFT.spin(directionType::fwd, speed, velocityUnits::pct);
00031     MOTOR_FRONT_RIGHT.spin(directionType::fwd, speed, velocityUnits::pct);
00032     MOTOR_BACK_RIGHT.spin(directionType::fwd, speed, velocityUnits::pct);
00033     MOTOR_BACK_LEFT.spin(directionType::fwd, speed, velocityUnits::pct);
00034   }
00035 }
```

2.17.1.12 stackControl()

```
void stackControl ( )
```

Reads the controller's button inputs to initiate the stack mechanism's movement while the button is being pressed

Author

Michael Baraty

Date

11/9/2019

Definition at line 77 of file drive.cpp.

```
00077                     {
00078   if(buttonIsPressed(MASTER.ButtonX))
00079     moveStackForward();
00080   else if(buttonIsPressed(MASTER.ButtonA))
00081     moveStackBack();
00082   else{
00083     MOTOR_STACK.stop(brakeType::brake);
00084   }
00085 }
```

2.17.1.13 tankDrive()

```
void tankDrive ( )
```

Initiates the tank drive control configuration for the controller, with the left y axis for the left side and the right y axis for the right side

Author

Michael Baraty

Date

11/9/2019

Definition at line 52 of file drive.cpp.

```
00052                 {
00053   int l = SPEED_MULTIPLIER * (.7  * (pow(axisValue(
      MASTER.Axis3) / 9, 3) / 10));
00054   int r = SPEED_MULTIPLIER * (.7  * (pow(axisValue(
      MASTER.Axis2) / 9, 3) / 10));
00055   int speedLeft = abs(l) > THRESHOLD? l: 0;
00056   int speedRight = abs(r) > THRESHOLD? r: 0;
00057
00058   if(slowMode){
00059     setSideSpeed(DriveSide::LEFT, speedLeft / 3);
00060      setSideSpeed(DriveSide::RIGHT, speedRight / 3);
00061   } else {
00062     setSideSpeed(DriveSide::LEFT, speedLeft);
00063     setSideSpeed(DriveSide::RIGHT, speedRight);
00064   }
00065 }
```

2.17.2 Variable Documentation

2.17.2.1 slowMode

```
bool slowMode = false
```

Definition at line 5 of file drive.cpp.

2.18 drive.cpp

```
00001 #include "drive.h"
00002 #include "declarations.h"
00003 #include "auton.h"
00004
00005 bool slowMode = false;
00006
00007 void drive() {
00008   arcadeDrive();
00009   stackControl();
00010   armControl();
00011   intakeControl();
00012
00013   if(buttonIsPressed(MASTER.ButtonUp)){
00014     slowMode = false;
00015   } else if(buttonIsPressed(MASTER.ButtonDown)){
00016     slowMode = true;
00017   }
00018
00019 }
00020
00021 void setSideSpeed(DriveSide side, int speed) {
00022   if(side == DriveSide::LEFT) {
00023     MOTOR_BACK_LEFT.spin(directionType::fwd, speed, velocityUnits::pct);
00024     MOTOR_FRONT_LEFT.spin(directionType::fwd, speed, velocityUnits::pct);
00025   } else if (side == DriveSide::RIGHT) {
00026     MOTOR_BACK_RIGHT.spin(directionType::fwd, speed, velocityUnits::pct);
00027     MOTOR_FRONT_RIGHT.spin(directionType::fwd, speed, velocityUnits::pct);
00028   }
00029   else {
```

```
00030     MOTOR_FRONT_LEFT.spin(directionType::fwd, speed, velocityUnits::pct);
00031     MOTOR_FRONT_RIGHT.spin(directionType::fwd, speed, velocityUnits::pct);
00032     MOTOR_BACK_RIGHT.spin(directionType::fwd, speed, velocityUnits::pct);
00033     MOTOR_BACK_LEFT.spin(directionType::fwd, speed, velocityUnits::pct);
00034   }
00035 }
00036
00037 void arcadeDrive() {
00038   int x = SPEED_MULTIPLIER  * -axisValue(MASTER.Axis1);
00039   int y = SPEED_MULTIPLIER * (.7  * (pow(-axisValue(
      MASTER.Axis3) / 9, 3) / 10));
00040   int speedLeft = abs(x + y) > THRESHOLD? -(x + y): 0;
00041   int speedRight = abs(x - y) > THRESHOLD? (x - y): 0;
00042
00043   if(slowMode){
00044     setSideSpeed(DriveSide::LEFT, speedLeft / 3);
00045      setSideSpeed(DriveSide::RIGHT, speedRight / 3);
00046   } else {
00047     setSideSpeed(DriveSide::LEFT, speedLeft);
00048     setSideSpeed(DriveSide::RIGHT, speedRight);
00049   }
00050 }
00051
00052 void tankDrive() {
00053   int l = SPEED_MULTIPLIER * (.7  * (pow(axisValue(
      MASTER.Axis3) / 9, 3) / 10));
00054   int r = SPEED_MULTIPLIER * (.7  * (pow(axisValue(
      MASTER.Axis2) / 9, 3) / 10));
00055   int speedLeft = abs(l) > THRESHOLD? l: 0;
00056   int speedRight = abs(r) > THRESHOLD? r: 0;
00057
00058   if(slowMode){
00059     setSideSpeed(DriveSide::LEFT, speedLeft / 3);
00060      setSideSpeed(DriveSide::RIGHT, speedRight / 3);
00061   } else {
00062     setSideSpeed(DriveSide::LEFT, speedLeft);
00063     setSideSpeed(DriveSide::RIGHT, speedRight);
00064   }
00065 }
00066
00067 void moveStackForward() {
00068   double final = 1.5;
00069   MOTOR_STACK.startSpinTo(10, rotationUnits::rev, 30, velocityUnits::pct);
00070 }
00071
00072 void moveStackBack() {
00073   double final = 0;
00074   MOTOR_STACK.startSpinTo(-10, rotationUnits::rev, 80, velocityUnits::pct);
00075 }
00076
00077 void stackControl() {
00078   if(buttonIsPressed(MASTER.ButtonX))
00079     moveStackForward();
00080   else if(buttonIsPressed(MASTER.ButtonA))
00081     moveStackBack();
00082   else{
00083     MOTOR_STACK.stop(brakeType::brake);
00084   }
00085 }
00086
00087 void armUp(){
00088   MOTOR_ARM.startSpinTo(60.6, rotationUnits::rev, 100, velocityUnits::pct);
00089   //realValue 6.6
00090 }
00091
00092 void armDown(){
00093   MOTOR_ARM.startSpinTo(-10, rotationUnits::rev, 100, velocityUnits::pct);
00094   //realValue 0
00095 }
00096
00097 void armControl(){
00098   if(buttonIsPressed(MASTER.ButtonR1)){
00099     armUp();
00100   }
00101   else if (buttonIsPressed(MASTER.ButtonR2)) {
00102     armDown();
00103   }
00104   else {
00105     MOTOR_ARM.stop(brakeType::hold);
00106   }
00107 }
```

```
00108
00109 void intakeIn() {
00110   if(!slowMode) {
00111     MOTOR_INTAKE_A.spin(directionType::fwd, 100, velocityUnits::pct);
00112     MOTOR_INTAKE_B.spin(directionType::fwd, 100, velocityUnits::pct);
00113   } else {
00114     MOTOR_INTAKE_A.spin(directionType::fwd, (100), velocityUnits::pct);
00115     MOTOR_INTAKE_B.spin(directionType::fwd, (100), velocityUnits::pct);
00116   }
00117 }
00118
00119 void intakeOut() {
00120    if(!slowMode) {
00121     MOTOR_INTAKE_A.spin(directionType::rev, 100, velocityUnits::pct);
00122     MOTOR_INTAKE_B.spin(directionType::rev, 100, velocityUnits::pct);
00123   } else {
00124     MOTOR_INTAKE_A.spin(directionType::rev, .5*(100), velocityUnits::pct);
00125     MOTOR_INTAKE_B.spin(directionType::rev, .5*(100), velocityUnits::pct);
00126   }
00127 }
00128
00129 void intakeControl() {
00130   if(buttonIsPressed(MASTER.ButtonL1)){
00131     intakeIn();
00132   }
00133   else if (buttonIsPressed(MASTER.ButtonL2)) {
00134     intakeOut();
00135   }
00136   else {
00137     MOTOR_INTAKE_A.stop(brakeType::hold);
00138     MOTOR_INTAKE_B.stop(brakeType::hold);
00139   }
00140 }
00141
00142
```

2.19 src/init.cpp File Reference

```
#include "init.h"
```

2.20 init.cpp

```
00001 #include "init.h"
```

2.21 src/main.cpp File Reference

```
#include "vex.h"
#include "drive.h"
#include "auton.h"
```

Functions

- int printDisplay ()
- void pre_auton (void)
- void autonomous (void)
- void usercontrol (void)
- int main ()

Variables

- vex::competition Competition
- vex::brain Brain
- controller MASTER = controller()
- task printTask
- motor MOTOR_BACK_LEFT = motor(PORT9, false)
- motor MOTOR_BACK_RIGHT = motor(PORT3, true)
- motor MOTOR_FRONT_LEFT = motor(PORT10, false)
- motor MOTOR_FRONT_RIGHT = motor(PORT2, true)
- motor MOTOR_INTAKE_A = motor(PORT15, gearSetting::ratio36_1, true)
- motor MOTOR_INTAKE_B = motor(PORT16, gearSetting::ratio36_1, false)
- motor MOTOR_STACK = motor(PORT17, false)
- motor MOTOR_ARM = motor(PORT12, gearSetting::ratio36_1, true)
- int i = 0

2.21.1 Function Documentation

2.21.1.1 autonomous()

```
void autonomous (
             void  )
```

Definition at line 80 of file main.cpp.

```
00080                         {
00081
00082   //printDisplay();
00083
00084   auton(Side::LEFT, Color::RED);
00085
00086   // ..........................................................................
00087   // Insert autonomous user code here.
00088   // ..........................................................................
00089
00090 }
```

2.21.1.2 main()

```
int main ( )
```

Definition at line 119 of file main.cpp.

```
00119           {
00120     //Set up callbacks for autonomous and driver control periods.
00121     Competition.autonomous( autonomous );
00122     Competition.drivercontrol( usercontrol );
00123
00124     //Run the pre-autonomous function.
00125     pre_auton();
00126
00127
00128     //Prevent main from exiting with an infinite loop.
00129     while(1) {
00130       printTask = task(printDisplay);
00131       vex::task::sleep(100);//Sleep the task for a short amount of time to prevent wasted resources.
00132     }
00133
00134 }
```

2.21.1.3 pre_auton()

```
void pre_auton (
              void  )
```

Definition at line 64 of file main.cpp.

```
00064                       {
00065   // All activities that occur before the competition starts
00066   // Example: clearing encoders, setting servo positions, ...
00067
00068 }
```

2.21.1.4 printDisplay()

```
int printDisplay ( )
```

Definition at line 35 of file main.cpp.

```
00035                     {
00036   while(true){
00037     Brain.Screen.printAt(0, 20, "%2.2f\n..>BACK LEFT", MOTOR_BACK_LEFT.temperature(
      temperatureUnits::celsius));
00038     Brain.Screen.printAt(0, 50, "%2.2f\n..>BACK RIGHT", MOTOR_BACK_RIGHT.temperature(
      temperatureUnits::celsius));
00039     Brain.Screen.printAt(0, 80, "%2.2f\n..>FRONT LEFT", MOTOR_FRONT_LEFT.temperature(
      temperatureUnits::celsius));
00040     Brain.Screen.printAt(0, 110, "%2.2f\n..>FRONT RIGHT", MOTOR_FRONT_RIGHT.
      temperature(temperatureUnits::celsius));
00041     Brain.Screen.printAt(0, 140, "%2.2f\n..>INTAKE A", MOTOR_INTAKE_A.temperature(
      temperatureUnits::celsius));
00042     Brain.Screen.printAt(0, 170, "%2.2f\n..>INTAKE B", MOTOR_INTAKE_B.temperature(
      temperatureUnits::celsius));
00043     Brain.Screen.printAt(0, 200, "%2.2f\n..>ARM", MOTOR_ARM.temperature(
      temperatureUnits::celsius));
00044     Brain.Screen.printAt(0, 230, "%2.2f\n..>MAGAZINE", MOTOR_STACK.temperature(
      temperatureUnits::celsius));
00045
00046   printf("%d\n", i);
00047   i++;
00048
00049     task::sleep(1000);
00050   }
00051 }
```

2.21.1.5 usercontrol()

```
void usercontrol (
              void  )
```

Definition at line 102 of file main.cpp.

```
00102                         {
00103   // User control code here, inside the loop
00104
00105   //auton(Side::RIGHT, Color::BLUE);
00106   while (1) {
00107
00108
00109     drive();
00110
00111     vex::task::sleep(20); //Sleep the task for a short amount of time to prevent wasted resources.
00112   }
00113 }
```

2.21.2 Variable Documentation

2.21.2.1 Brain

```
vex::brain Brain
```

Definition at line 17 of file main.cpp.

2.21.2.2 Competition

```
vex::competition Competition
```

Definition at line 16 of file main.cpp.

2.21.2.3 i

```
int i = 0
```

Definition at line 33 of file main.cpp.

2.21.2.4 MASTER

```
controller MASTER = controller()
```

Makes the main controller accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 18 of file main.cpp.

2.21.2.5 MOTOR_ARM

```
motor MOTOR_ARM = motor(PORT12, gearSetting::ratio36_1, true)
```

Makes the intake lifter motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 29 of file main.cpp.

2.21.2.6 MOTOR_BACK_LEFT

```
motor MOTOR_BACK_LEFT = motor(PORT9, false)
```

Makes the back left motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 22 of file main.cpp.

2.21.2.7 MOTOR_BACK_RIGHT

```
motor MOTOR_BACK_RIGHT = motor(PORT3, true)
```

Makes the back right motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 23 of file main.cpp.

2.21.2.8 MOTOR_FRONT_LEFT

```
motor MOTOR_FRONT_LEFT = motor(PORT10, false)
```

Makes the front left motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 24 of file main.cpp.

2.21.2.9 MOTOR_FRONT_RIGHT

```
motor MOTOR_FRONT_RIGHT = motor(PORT2, true)
```

Makes the front right motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 25 of file main.cpp.

2.21.2.10 MOTOR_INTAKE_A

```
motor MOTOR_INTAKE_A = motor(PORT15, gearSetting::ratio36_1, true)
```

Makes the right intake motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 26 of file main.cpp.

2.21.2.11 MOTOR_INTAKE_B

```
motor MOTOR_INTAKE_B = motor(PORT16, gearSetting::ratio36_1, false)
```

Makes the left intake motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 27 of file main.cpp.

2.21.2.12 MOTOR_STACK

```
motor MOTOR_STACK = motor(PORT17, false)
```

Makes the stack mechanism motor accessible in other files than main.cpp

Author

Michael Baraty

Date

11/9/2019

Definition at line 28 of file main.cpp.

2.21.2.13 printTask

```
task printTask
```

Definition at line 20 of file main.cpp.

2.22 main.cpp

```
00001 /*----------------------------------------------------------------------------*/
00002 /*                                                                            */
00003 /*    Module:       main.cpp                                                  */
00004 /*    Author:       mbaraty                                                   */
00005 /*    Created:      Thu Sep 12 2019                                           */
00006 /*    Description:  V5 project                                                */
00007 /*                                                                            */
00008 /*----------------------------------------------------------------------------*/
00009 #include "vex.h"
00010 #include "drive.h"
00011 #include "auton.h"
00012
00013 using namespace vex;
00014
00015 // A global instance of vex::competition
00016 vex::competition Competition;
00017 vex::brain Brain;
00018 controller MASTER = controller();
00019
00020 task printTask;
00021
00022 motor MOTOR_BACK_LEFT   = motor(PORT9, false);
00023 motor MOTOR_BACK_RIGHT  = motor(PORT3, true);
00024 motor MOTOR_FRONT_LEFT  = motor(PORT10, false);
00025 motor MOTOR_FRONT_RIGHT = motor(PORT2, true);
00026 motor MOTOR_INTAKE_A    = motor(PORT15, gearSetting::ratio36_1, true);
00027 motor MOTOR_INTAKE_B    = motor(PORT16, gearSetting::ratio36_1, false);
00028 motor MOTOR_STACK       = motor(PORT17, false);
00029 motor MOTOR_ARM         = motor(PORT12, gearSetting::ratio36_1, true);
00030
00031 // define your global instances of motors and other devices here
00032
00033 int i = 0;
00034
00035 int printDisplay() {
00036   while(true){
00037     Brain.Screen.printAt(0, 20, "%2.2f\n..>BACK LEFT", MOTOR_BACK_LEFT.temperature(
      temperatureUnits::celsius));
00038     Brain.Screen.printAt(0, 50, "%2.2f\n..>BACK RIGHT", MOTOR_BACK_RIGHT.temperature(
      temperatureUnits::celsius));
00039     Brain.Screen.printAt(0, 80, "%2.2f\n..>FRONT LEFT", MOTOR_FRONT_LEFT.temperature(
      temperatureUnits::celsius));
00040     Brain.Screen.printAt(0, 110, "%2.2f\n..>FRONT RIGHT", MOTOR_FRONT_RIGHT.
      temperature(temperatureUnits::celsius));
00041     Brain.Screen.printAt(0, 140, "%2.2f\n..>INTAKE A", MOTOR_INTAKE_A.temperature(
      temperatureUnits::celsius));
00042     Brain.Screen.printAt(0, 170, "%2.2f\n..>INTAKE B", MOTOR_INTAKE_B.temperature(
      temperatureUnits::celsius));
00043     Brain.Screen.printAt(0, 200, "%2.2f\n..>ARM", MOTOR_ARM.temperature(
      temperatureUnits::celsius));
00044     Brain.Screen.printAt(0, 230, "%2.2f\n..>MAGAZINE", MOTOR_STACK.temperature(
      temperatureUnits::celsius));
00045
00046   printf("%d\n", i);
00047   i++;
00048
00049     task::sleep(1000);
00050   }
00051 }
00052
00053
00054 /*----------------------------------------------------------------------------*/
00055 /*                           Pre-Autonomous Functions                         */
00056 /*                                                                            */
00057 /*  You may want to perform some actions before the competition starts.       */
00058 /*  Do them in the following function.  You must return from this function    */
00059 /*  or the autonomous and usercontrol tasks will not be started.  This        */
00060 /*  function is only called once after the cortex has been powered on and     */
00061 /*  not every time that the robot is disabled.                                */
00062 /*----------------------------------------------------------------------------*/
00063
00064 void pre_auton( void ) {
00065   // All activities that occur before the competition starts
00066   // Example: clearing encoders, setting servo positions, ...
00067
00068 }
00069
00070 /*----------------------------------------------------------------------------*/
```

```
00071 /*                                                                           */
00072 /*                              Autonomous Task                              */
00073 /*                                                                           */
00074 /*  This task is used to control your robot during the autonomous phase of   */
00075 /*  a VEX Competition.                                                       */
00076 /*                                                                           */
00077 /*  You must modify the code to add your own robot specific commands here.   */
00078 /*---------------------------------------------------------------------------*/
00079
00080 void autonomous( void ) {
00081
00082   //printDisplay();
00083
00084   auton(Side::LEFT, Color::RED);
00085
00086   // ..........................................................................
00087   // Insert autonomous user code here.
00088   // ..........................................................................
00089
00090 }
00091
00092 /*---------------------------------------------------------------------------*/
00093 /*                                                                           */
00094 /*                              User Control Task                            */
00095 /*                                                                           */
00096 /*  This task is used to control your robot during the user control phase of */
00097 /*  a VEX Competition.                                                       */
00098 /*                                                                           */
00099 /*  You must modify the code to add your own robot specific commands here.   */
00100 /*---------------------------------------------------------------------------*/
00101
00102 void usercontrol( void ) {
00103   // User control code here, inside the loop
00104
00105   //auton(Side::RIGHT, Color::BLUE);
00106   while (1) {
00107
00108
00109     drive();
00110
00111     vex::task::sleep(20); //Sleep the task for a short amount of time to prevent wasted resources.
00112   }
00113 }
00114
00115
00116 //
00117 // Main will set up the competition functions and callbacks.
00118 //
00119 int main() {
00120     //Set up callbacks for autonomous and driver control periods.
00121     Competition.autonomous( autonomous );
00122     Competition.drivercontrol( usercontrol );
00123
00124     //Run the pre-autonomous function.
00125     pre_auton();
00126
00127
00128     //Prevent main from exiting with an infinite loop.
00129     while(1) {
00130       printTask = task(printDisplay);
00131       vex::task::sleep(100);//Sleep the task for a short amount of time to prevent wasted resources.
00132     }
00133
00134 }
```