# Header Files

## include/auton.hpp

1 *#pragma once*

2 *#include "main.h"*

3

4 */\*\**

5 *\* @brief Originally, our plan was to use an auton selector, but due to*

6 *\* time constraints and bugs, we were unable to finish it on time.*

7 *\**

8 *\*/*

9

10 *//-----------------------------------------------------------//*

11

12 */\*\**

13 *\* @brief Generates the trajectories used in skills autnonomous*

14 *\*/*

15 void genSkills();

16

17 */\*\**

18 *\* @brief Executes skills autonomous*

19 *\*/*

20 void skills();

21

22 */\*\**

23 *\* @brief Generates the trajectories used in red alliance's left side autnonomous*

24 *\*/*

25 void genRedLeft();

26

27 */\*\**

28 *\* @brief Executes red alliance's left side autonomous*

29 *\*/*

30 void redLeft();

31

32 */\*\**

33 *\* @brief Generates the trajectories used in red alliance's right side autnonomous*

34 *\*/*

35 void genRedRight();

36

37 */\*\**

38 *\* @brief Executes red alliance's right side autonomous*

39 *\*/*

40 void redRight();

41

42 */\*\**

43 *\* @brief Generates the trajectories used in blue alliance's left side autnonomous*

44 *\*/*

45 void genBlueLeft();

46

47 */\*\**

48 *\* @brief Executes blue alliance's left side autonomous*

49 *\*/*

50 void blueLeft();

51

52 */\*\**

53 *\* @brief Generates the trajectories used in blue alliance's right side autnonomous*

54 *\*/*

55 void genBlueRight();

56

57 */\*\**

58 *\* @brief Executes blue alliance's right side autonomous*

59 *\*/*

60 void blueRight();

61

62 */\*\**

63 *\* @brief Generates the trajectories used in AWP autnonomous*

64 *\*/*

65 void genAwp();

66

67 */\*\**

68 *\* @brief Executes awp autonomous*

69 *\*/*

70 void awp();

## include/drive.hpp

1 *#pragma once*

2 *#include "main.h"*

3

4 */\*\**

5 *\* @brief Computes the desired drive speed to control the chassis using curvature control*

6 *\* The turn power is scaled with the linear power in order to allow finer control*

7 *\* of the chassis.*

8 *\**

9 *\* @param moveC - the desired linear speed for the robot to move in, normalized to [-1,*

*C→ 1]*

10 *\* @param turnC - the desired curvature for the robot to move in, normalized to [-1, 1]*

11 *\* @param quickTurn - when this parameter is set to true, arcade control is used instead*

*C→*

12 *\**

*to allow point turn*

13 *\* @return the desired left and right velocity for the chassis motors to move in,*

*C→*

14 *\*/*

*normalized to [-1, 1]*

15 std::pair<double, double> curvatureDrive(double moveC, double turnC, bool quickTurn);

16

17 */\*\**

18 *\* @brief Turns the robot to the desired global angle (using closed-loop control)*

19 *\**

20 *\* @param targetAngle - the target odometry global angle to turn to, normalized to*

*C→*

21 *\*/*

*[-2pi, 2pi]*

22 void turnToAngle(okapi::QAngle targetAngle);

23

24 */\*\**

25 *\* @brief This is our custom velocity controller. Although the motor has decent internal*

26 *\* PID velocity control, through our testing, we realized that it still is not*

27 *\* as precise as this custom velocity controller. Yet due to time constraints,*

28 *\* we did not have enough to time to individually tune our motors, and thus, we*

29 *\* stuck with using the motor's internal velocity controller.*

30 *\**

31 *\* The reason why we even have this is because our drive's movement in auton*

*C→ heavily*

32 *\* relies on 2D motion profiles. There is lots of complicated math behind it, but*

*C→ in*

33 *\* a sense, we create a smooth trajectory for our robot to follow and convert it*

*C→ to*

34 *\* actual drive velocity that our robot follows for every 10 ms. As a result, the*

35 *\* how precise our auton is heavily relies on how well the velocity control of*

*C→ our*

36 *\* motors are.*

37 *\**

38 *\* @param velocity desired velocity*

39 *\* @param accel desired acceleration*

40 *\* @param currSpeed current speed of the motor(s)*

41 *\* @return output for the motor(s)*

42 *\*/*

43 double velControl(QSpeed velocity, QAcceleration accel, QSpeed currSpeed);

44

45 */\*\**

46 *\* @brief The trajectory generator we use outputs linear velocity (ft/s). In order for*

*C→ us to*

47 *\* make use of that, we must convert the linear velocity to RPM, which the motors*

*C→ follow.*

48 *\* Although there are much more mathematical ways of making this conversion (such*

*C→ as*

49 *\* recording the wheel size, gear ratio, and motor speed, and using those values*

*C→ to*

50 *\* convert) we decided to use a much simpler method of calculating ratios. Since*

*C→ the*

51 *\* maximum motor RPM (for our drive) is 600 RPM and the maximum linear velocity*

*C→ is*

52 *\* 4.92126 ft/s, we can use simple ratios to convert the desired linear velocity*

*C→*

53 *\**

*to RPM.*

54 *\* @param path 2D vector of linear velocity*

55 *\* @return vector of RPM*

56 *\*/*

57 std::vector<double> pathToRPM(std::vector<std::vector<double>> path);

58

59 */\*\**

60 *\* @brief This is the function which actually makes the robot drive. followPath receives*

*C→ inputs*

61 *\* of the desired left and right velocites and feeds it to the drive motor to*

*C→ follow*

62 *\* the desired path.*

63 *\**

64 *\* @param leftVel 2D vector of left velocity*

65 *\* @param rightVel 2D vector of right velocity*

66 *\*/*

67 void followPath(std::vector<std::vector<double>> leftVel,

*C→* std::vector<std::vector<double>> rightVel);

## include/globals.hpp

1 *#pragma once*

2 *#include "main.h"*

3

4 */\*\**

5 *\* @brief set 'Pneumatic' as pros::ADIDigitalOut*

6 *\**

7 *\*/*

8 using Pneumatic = pros::ADIDigitalOut;

9

10 *// CONSTANTS*

11 */\*\**

12 *\* @brief Here is our constants (though we only ended up really using*

13 *\* DEADBAND since we had no time to tune lift)*

14 *\**

15 *\*/*

16 const double DEADBAND = 0.0500;

17 const double MAXLIFTHEIGHT = 2000;

18 const double LIFT\_INCREMENT = 10;

19

20 *// CONTROLLER*

21 */\*\**

22 *\* @brief Our controller*

23 *\**

24 *\*/*

25 extern Controller master;

26

27 *// MOTORS*

28 */\*\**

29 *\* @brief These are all the motors we used*

30 *\* (each of our drive motor groups have 3 individual motors)*

31 *\**

32 *\*/*

33 extern MotorGroup leftDrive;

34 extern MotorGroup rightDrive;

35 extern pros::Motor lift;

36 extern Motor roller;

37

38 *// SENSORS*

39 */\*\**

40 *\* @brief These were all sensors we were planning on using. But due to*

41 *\* time constraints, mechanical issues, and our insanely accurate*

42 *\* pathing algorithm, we found no need to use these sensors.*

43 *\**

44 *\*/*

45 extern ADIEncoder trackLeft;

46 extern ADIEncoder trackRight;

47 extern ADIEncoder trackMiddle;

48 extern RotationSensor liftSensor;

49 extern IMU imu;

50

51 *// PNEUMATICS*

52 */\*\**

53 *\* @brief Globally declares our solenoids (one for claw, one for back mogo lift)*

54 *\**

55 *\*/*

56 extern Pneumatic mogo;

57 extern Pneumatic claw;

58

59 *// SUBSYSTEM CONTROLLERS*

60 */\*\**

61 *\* @brief All of the following are controllers from Okpailib. But since we were*

62 *\* quite sucessful with our own custom path follower, we decided not to*

63 *\* use these.*

64 *\**

65 *\*/*

66 extern std::shared\_ptr<ChassisController> chassis;

67 extern std::shared\_ptr<AsyncMotionProfileController> profiler;

68 extern std::shared\_ptr<AsyncPositionController<double, double>> liftController;

69 extern std::shared\_ptr<IterativePosPIDController> turnPID;

70

71

72 *// AUTONOMOUS CONTROLLER*

73 */\*\**

74 *\* @brief As stated previously, due to time constraints and bugs, we were not able*

75 *\* able to have our auton selector ready in time.*

76 *\**

77 *\*/*

78 extern int selectedAuton;

79 extern std::map<int, std::function<void()>> auton;

80 extern std::map<int, std::function<void()>> path;

## include/main.h

1 */\*\**

2 *\* \file main.h*

3 *\**

4 *\* Contains common definitions and header files used throughout your PROS*

5 *\* project.*

6 *\**

7 *\* Copyright (c) 2017-2021, Purdue University ACM SIGBots.*

8 *\* All rights reserved.*

9 *\**

10 *\* This Source Code Form is subject to the terms of the Mozilla Public*

11 *\* License, v. 2.0. If a copy of the MPL was not distributed with this*

12 *\* file, You can obtain one at* [*http://mozilla.org/MPL/2.0/.*](http://mozilla.org/MPL/2.0/)

13 *\*/*

14

15 *#ifndef \_PROS\_MAIN\_H\_*

16 *#define \_PROS\_MAIN\_H\_*

17

18 */\*\**

19 *\* If defined, some commonly used enums will have preprocessor macros which give*

20 *\* a shorter, more convenient naming pattern. If this isn't desired, simply*

21 *\* comment the following line out.*

22 *\**

23 *\* For instance, E\_CONTROLLER\_MASTER has a shorter name: CONTROLLER\_MASTER.*

24 *\* E\_CONTROLLER\_MASTER is pedantically correct within the PROS styleguide, but*

25 *\* not convienent for most student programmers.*

26 *\*/*

27 *#define PROS\_USE\_SIMPLE\_NAMES*

28

29 */\*\**

30 *\* If defined, C++ literals will be available for use. All literals are in the*

31 *\* pros::literals namespace.*

32 *\**

33 *\* For instance, you can do `4\_mtr = 50` to set motor 4's target velocity to 50*

34 *\*/*

35 *#define PROS\_USE\_LITERALS*

36

37 *#include "api.h"*

38

39 */\*\**

40 *\* You should add more #includes here*

41 *\*/*

42 *#include "okapi/api.hpp"*

43 *#include "pros/api\_legacy.h"*

44 *#include "pros/apix.h"*

45

46 */\*\**

47 *\* If you find doing pros::Motor() to be tedious and you'd prefer just to do*

48 *\* Motor, you can use the namespace with the following commented out line.*

49 *\**

50 *\* IMPORTANT: Only the okapi or pros namespace may be used, not both*

51 *\* concurrently! The okapi namespace will export all symbols inside the pros*

52 *\* namespace.*

53 *\*/*

54 *// using namespace pros;*

55 *// using namespace pros::literals;*

56 using namespace okapi;

57

58 */\*\**

59 *\* Prototypes for the competition control tasks are redefined here to ensure*

60 *\* that they can be called from user code (i.e. calling autonomous from a*

61 *\* button press in opcontrol() for testing purposes).*

62 *\*/*

63 *#ifdef cplusplus*

64 extern "C" {

65 *#endif*

66 void autonomous(void);

67 void initialize(void);

68 void disabled(void);

69 void competition\_initialize(void);

70 void opcontrol(void);

71 *#ifdef cplusplus*

72 }

73 *#endif*

74

75 *#ifdef cplusplus*

76 */\*\**

77 *\* You can add C++-only headers here*

78 *\*/*

79 *#include <iostream>*

80 *#include <cmath>*

81 *#include <algorithm>*

82 *#include <map>*

83 *#include "globals.hpp"*

84 *#include "drive.hpp"*

85 *#include "auton.hpp"*

86 *#include "paths/leftPaths.hpp"*

87 *#include "paths/awp.hpp"*

88 *#include "paths/rightPaths.hpp"*

89 *#include "gif-pros/gifclass.hpp"*

90

91

92 *#endif*

93

94 *#endif // \_PROS\_MAIN\_H\_*

## include/gif-pros/gifclass.hpp

1 *#pragma once*

2 *#include "main.h"*

3 *#include "gifdec.h"*

4

5 */\*\**

6 *\* MIT License*

7 *\* Copyright (c) 2019 Theo Lemay*

8 *\* https://github.com/theol0403/gif-pros*

9 *\*/*

10

11 class Gif {

12

13 public:

14

15 */\*\**

16 *\* Construct the Gif class*

17 *\* @param fname the gif filename on the SD card (prefixed with /usd/)*

18 *\* @param parent the LVGL parent object*

19 *\*/*

20 Gif(const char\* fname, lv\_obj\_t\* parent);

21

22 */\*\**

23 *\* Destructs and cleans the Gif class*

24 *\*/*

25 ~Gif();

26

27 */\*\**

28 *\* Pauses the GIF task*

29 *\*/*

30 void pause();

31

32 */\*\**

33 *\* Resumes the GIF task*

34 *\*/*

35 void resume();

36

37 */\*\**

38 *\* Deletes GIF and frees all allocated memory*

39 *\*/*

40 void clean();

41

42 private:

43

44 gd\_GIF\* \_gif = nullptr; *// gif decoder object*

45 void\* \_gifmem = nullptr; *// gif file loaded from SD into memory*

46 uint8\_t\* \_buffer = nullptr; *// decoder frame buffer*

47

48 lv\_color\_t\* \_cbuf = nullptr; *// canvas buffer*

49 lv\_obj\_t\* \_canvas = nullptr; *// canvas object*

50

51 pros::task\_t \_task = nullptr; *// render task*

52

53 */\*\**

54 *\* Cleans and frees all allocated memory*

55 *\*/*

56 void \_cleanup();

57

58 */\*\**

59 *\* Render cycle, blocks until loop count exceeds gif loop count flag (if any)*

60 *\*/*

61 void \_render();

62

63 */\*\**

64 *\* Calls \_render()*

65 *\* @param arg Gif\**

66 *\*/*

67 static void \_render\_task(void\* arg);

68

69 };

## include/gif-pros/gifdec.h

1 *#ifndef GIFDEC\_H*

2 *#define GIFDEC\_H*

3

4 *#include <stdio.h>*

5 *#include <stdint.h>*

6 *#include <sys/types.h>*

7

8 *#ifdef cplusplus*

9 extern "C" {

10 *#endif*

11

12 *#define BYTES\_PER\_PIXEL 4*

13

14 typedef struct gd\_Palette {

15 int size;

16 uint8\_t colors[0x100 \* 3];

17 } gd\_Palette;

18

19 typedef struct gd\_GCE {

20 uint16\_t delay;

21 uint8\_t tindex;

22 uint8\_t disposal;

23 int input;

24 int transparency;

25 } gd\_GCE;

26

27 typedef struct gd\_GIF {

28 FILE \*fp;

29 off\_t anim\_start;

30 uint16\_t width, height;

31 uint16\_t depth;

32 uint16\_t loop\_count;

33 gd\_GCE gce;

34 gd\_Palette \*palette;

35 gd\_Palette lct, gct;

36 void (\*plain\_text)(

37 struct gd\_GIF \*gif, uint16\_t tx, uint16\_t ty,

38 uint16\_t tw, uint16\_t th, uint8\_t cw, uint8\_t ch,

39 uint8\_t fg, uint8\_t bg

40 );

41 void (\*comment)(struct gd\_GIF \*gif);

42 void (\*application)(struct gd\_GIF \*gif, char id[8], char auth[3]);

43 uint16\_t fx, fy, fw, fh;

44 uint8\_t bgindex;

45 uint8\_t \*canvas, \*frame;

46 } gd\_GIF;

47

48 gd\_GIF \*gd\_open\_gif(FILE \*fp);

49 int gd\_get\_frame(gd\_GIF \*gif);

50 void gd\_render\_frame(gd\_GIF \*gif, uint8\_t \*buffer);

51 void gd\_rewind(gd\_GIF \*gif);

52 void gd\_close\_gif(gd\_GIF \*gif);

53

54 *#ifdef cplusplus*

55 }

56 *#endif*

57

58 *#endif /\* GIFDEC\_H \*/*

## include/paths/awp.hpp

1 *#pragma once*

2 *#include "main.h"*

3

4 */\*\**

5 *\* @brief The follow class contains all paths used in the auton win point (AWP) auton*

6 *\**

7 *\*/*

8 class AWP {

9 public:

10 static std::vector<std::vector<double>> fwdRLeft;

11 static std::vector<std::vector<double>> fwdRRight;

12

13 static std::vector<std::vector<double>> loopbackLeft;

14 static std::vector<std::vector<double>> loopbackRight;

15 };

## include/paths/leftPaths.hpp

1 *#pragma once*

2 *#include "main.h"*

3

4 */\*\**

5 *\* @brief The following class contains all paths used in the left auton.*

6 *\* In the auton, we first capture the left neutral mogo, score a ring*

7 *\* and bring it back. As we head back to our home row, we move to our*

8 *\* alliance mogo where we deposit a ring into the base of our mogo.*

9 *\**

10 *\*/*

11 class LeftPaths {

12 public:

13 static std::vector<std::vector<double>> pathLeft;

14 static std::vector<std::vector<double>> pathRight;

15

16 static std::vector<std::vector<double>> pathLeftR;

17 static std::vector<std::vector<double>> pathRightR;

18

19 static std::vector<std::vector<double>> fwdLeft;

20 static std::vector<std::vector<double>> fwdRight;

21 };

## include/paths/rightPaths.hpp

1 *#pragma once*

2 *#include "main.h"*

3

4 */\*\**

5 *\* @brief The following class contains all paths used in the right auton.*

6 *\* The right auton begins by curving to the tall neutral mogo, tipping*

7 *\* it over, and going back to score 2 rings on our right alliance mogo.*

8 *\**

9 *\*/*

10 class RightPaths {

11 public:

12 static std::vector<std::vector<double>> curveLeft;

13 static std::vector<std::vector<double>> curveRight;

14

15 static std::vector<std::vector<double>> curveRLeft;

16 static std::vector<std::vector<double>> curveRRight;

17

18 static std::vector<std::vector<double>> fwdLeft;

19 static std::vector<std::vector<double>> fwdRight;

20 };

# Source Files

## src/auton.cpp

1 */\*\**

2 *\* @brief Unfortunately due to time constraints and bugs, we were never*

3 *\* able to finish our auton selector/organizational system*

4 *\**

5 *\*/*

6

7 *// #include "main.h"*

8

9 *// void genSkills(){}*

10

11 *// void skills(){*

12 *// chassis->setState({0\_in, 0\_in, 0\_deg});*

13 *// }*

14

15 *// void genRedLeft(){*

16 *// profiler->generatePath({{0\_in, 0\_in, 0\_deg}, {0\_in, 24\_in, 0\_deg}}, "Test");*

17 *// }*

18

19 *// void redLeft(){*

20 *// chassis->setState({0\_in, 0\_in, 0\_deg});*

21 *// profiler->setTarget("Test");*

22 *// profiler->waitUntilSettled();*

23

24 *// turnToAngle(90\_deg);*

25 *// }*

26

27 *// void genRedRight(){}*

28

29 *// void redRight(){*

30 *// chassis->setState({0\_in, 0\_in, 0\_deg});*

31 *// }*

32

33 *// void genBlueLeft(){}*

34

35 *// void blueLeft(){*

36 *// chassis->setState({0\_in, 0\_in, 0\_deg});*

37 *// }*

38

39 *// void genBlueRight(){}*

40

41 *// void blueRight(){*

42 *// chassis->setState({0\_in, 0\_in, 0\_deg});*

43 *// }*

44

45 *// void genAwp(){}*

46

47 *// void awp(){*

48 *// chassis->setState({0\_in, 0\_in, 0\_deg});*

49 *// }*

## src/drive.cpp

1 *#include "main.h"*

2

3 std::pair<double, double> curvatureDrive(double moveC, double turnC, bool quickTurn){

4 *// Compute velocity, right stick = curvature if no quickturn, else power*

5 double leftSpeed = moveC + (quickTurn ? turnC : abs(moveC) \* turnC);

6 double rightSpeed = moveC - (quickTurn ? turnC : abs(moveC) \* turnC);

7

8 *// Normalize velocity*

9 double maxMagnitude = std::max(abs(leftSpeed), abs(rightSpeed));

10 if (maxMagnitude > 1.0) {

11 leftSpeed /= maxMagnitude;

12 rightSpeed /= maxMagnitude;

13 }

14

15 return std::make\_pair(leftSpeed, rightSpeed);

16 }

17

18 double velControl(QSpeed velocity, QAcceleration accel, QSpeed currSpeed) {

19 double kV = 0.0, kA = 0.0, kP = 0.0;

20 return kV \* velocity.convert(okapi::mps) + kA \* accel.convert(okapi::mps2) + kP \*

21 }

*C→* (velocity - currSpeed).convert(okapi::mps);

22

23 *// velocity only, doesn't use custon velControl*

24 std::vector<double> pathToRPM(std::vector<std::vector<double>> path) {

25 std::vector<double> newPath;

26 double val = 0.0;

27 for(int i = 0; i < path.size(); i++) {

28 *// max vel = 4.92126 ft/s, max rpm = 600 --> convert values to rpm*

29 val = (path[i][0] \* 600) / 4.92126;

30 newPath.push\_back(val);

31 }

32 return newPath;

33 }

34

35 void followPath(std::vector<std::vector<double>> leftVel,

*C→* std::vector<std::vector<double>> rightVel) {

36 std::vector<double> left = pathToRPM(leftVel); std::vector<double> right =

*C→* pathToRPM(rightVel);

37 for(int i = 0; i < left.size() || i < right.size(); i++) {

38 leftDrive.moveVelocity(left[i]);

39 rightDrive.moveVelocity(right[i]);

40 pros::delay(10);

41 *// pros::Task::delay\_until(&now, 10);*

42 }

43 leftDrive.moveVelocity(0);

44 rightDrive.moveVelocity(0);

45 }

46

47 *// void turnToAngle(okapi::QAngle targetAngle){*

48 *// turnPID->reset();*

49 *// turnPID->setTarget(targetAngle.convert(degree));*

50

51 *// do {*

52 *// // chassis->getOdometry()->step();*

53 *// // double power = turnPID->step(chassis->getState().theta.convert(degree));*

54 *// (chassis->getModel())->tank(power, -power);*

55 *// pros::delay(10);*

56 *// } while(!turnPID->isSettled());*

57

58 *// (chassis->getModel())->stop();*

59 *// }*

## src/globals.cpp

1 *#include "main.h"*

2

3 *// CONTROLLER*

4 Controller master(ControllerId::master);

5

6 *// MOTORS*

7 Motor leftTop(4, false, AbstractMotor::gearset::blue,

*C→* AbstractMotor::encoderUnits::degrees);

8 Motor leftMiddle(5, true, AbstractMotor::gearset::blue,

*C→* AbstractMotor::encoderUnits::degrees);

9 Motor leftBottom(6, false, AbstractMotor::gearset::blue,

*C→* AbstractMotor::encoderUnits::degrees);

10 Motor rightTop(7, true, AbstractMotor::gearset::blue,

*C→* AbstractMotor::encoderUnits::degrees);

11 Motor rightMiddle(8, false, AbstractMotor::gearset::blue,

*C→* AbstractMotor::encoderUnits::degrees);

12 Motor rightBottom(9, true, AbstractMotor::gearset::blue,

*C→* AbstractMotor::encoderUnits::degrees);

13 MotorGroup leftDrive({leftTop, leftMiddle, leftBottom});

14 MotorGroup rightDrive({rightTop, rightMiddle, rightBottom});

15 *// Motor lift(10, true, AbstractMotor::gearset::green,*

*C→ AbstractMotor::encoderUnits::degrees);*

16 pros::Motor lift(10, true);

17 *// Motor roller(8, false, AbstractMotor::gearset::blue,*

*C→ AbstractMotor::encoderUnits::degrees); // TODO - Change Port*

18

19 *// SENSORS*

20 ADIEncoder trackLeft({14, 'A', 'B'}, false); *// TODO - Change Port, reverse?*

21 ADIEncoder trackRight({14, 'C', 'D'}, false); *// TODO - Change Port, reverse?*

22 ADIEncoder trackMiddle({14, 'E', 'F'}, false); *// TODO - Change Port, reverse?*

23 RotationSensor liftSensor(20, false); *// TODO - Change Port, reverse?*

24 IMU imu(14);

25

26 *// PNEUMATICS*

27 Pneumatic mogo('C');

28 Pneumatic claw('D'); *// TODO - Change Port*

29

30 *// SUBSYSTEM CONTROLLERS*

31 std::shared\_ptr<ChassisController> chassis = ChassisControllerBuilder()

32 .withMotors(leftDrive, rightDrive)

33 .withDimensions({AbstractMotor::gearset::blue, 5.0/3.0}, {{3.25\_in, 38.5\_cm},

*C→* imev5BlueTPR}) *// TODO - Change Track Width*

34 *// .withSensors(trackLeft, trackRight, trackMiddle)*

35 *// .withOdometry({{2.75\_in, 6.25\_in}, quadEncoderTPR}) // TODO - Change Track Width*

36 *// .withOdometry()*

37 *// .buildOdometry();*

38 .build();

39

40 std::shared\_ptr<AsyncMotionProfileController> profiler =

*C→* AsyncMotionProfileControllerBuilder()

41 .withLimits({ *// TODO - Tune Max Robot Velocity / Acceleration*

42 1.5, *// Maximum linear velocity of the Chassis in m/s*

|  |  |  |
| --- | --- | --- |
| 43 |  | 4.0, *// Maximum linear acceleration of the Chassis in m/s/s* |
| 44 |  | 6.0 *// Maximum linear jerk of the Chassis in m/s/s/s* |
| 45 |  | }) |
| 46 |  | .withOutput(chassis) |
| 47 |  | .buildMotionProfileController(); |
| 48 |  |  |
| 49 | *//* | *std::shared\_ptr<AsyncPositionController<double, double>> liftController =* |
| 50 | *C→*  *//* | *AsyncPosControllerBuilder()*  *.withMotor(lift)* |
| 51 | *//* | *.withGains({0.01, 0.001, 0.0000}) // TODO - Slightly tune constant* |
| 52 | *//* | *.withSensor(std::make\_shared<okapi::RotationSensor>(liftSensor))* |
| 53 | *//* | *.build();* |
| 54 |  |  |

55 std::shared\_ptr<IterativePosPIDController> turnPID =

*C→* std::make\_shared<IterativePosPIDController>(0, 0, 0, 0,

*C→* TimeUtilFactory::withSettledUtilParams(2, 2, 100\_ms)); *// #TODO - Tune Constant*

56

57

58 *// AUTONOMOUS CONTROLLER*

59 int selectedAuton = 1;

60 std::map<int, std::function<void()>> auton;

61 std::map<int, std::function<void()>> path;

## src/main.cpp

1 *#include "main.h"*

2

3 void initialize(){

4 pros::lcd::initialize();

5 master.setText(0, 0, "Current Autonomous: " + std::to\_string(selectedAuton));

6 pros::lcd::set\_text(4, "init");

7

8

9 *// Initializes Controller*

10 *// liftController->tarePosition();*

11

12 *// Adds autonomous*

13 *// auton.insert(std::make\_pair(0, [](){})); // lambda ftw*

14 *// auton.insert(std::make\_pair(1, redLeft));*

15 *// auton.insert(std::make\_pair(2, redRight));*

16 *// auton.insert(std::make\_pair(3, blueLeft));*

17 *// auton.insert(std::make\_pair(4, blueRight));*

18 *// auton.insert(std::make\_pair(5, awp));*

19 *// auton.insert(std::make\_pair(6, skills));*

20

21 *// // Adds path generation functions*

22 *// path.insert(std::make\_pair(0, [](){}));*

23 *// path.insert(std::make\_pair(1, genRedLeft));*

24 *// path.insert(std::make\_pair(2, genRedRight));*

25 *// path.insert(std::make\_pair(3, genBlueLeft));*

26 *// path.insert(std::make\_pair(4, genBlueRight));*

27 *// path.insert(std::make\_pair(5, genAwp));*

28 *// path.insert(std::make\_pair(6, genSkills));*

29

30 *// // Generates path based on pre-selected auton*

31 *// path[selectedAuton]();*

32 }

33

34 void disabled(){}

35

36 void competition\_initialize(){}

37

38 void autonomous(){

39 *// INITIALIZATION*

40 lift.set\_brake\_mode(pros::motor\_brake\_mode\_e::E\_MOTOR\_BRAKE\_HOLD);

41 leftDrive.setBrakeMode(AbstractMotor::brakeMode::hold);

42 rightDrive.setBrakeMode(AbstractMotor::brakeMode::hold);

43

44 *//----------------------------------------------------------------------//*

45 *// LEFT AUTON*

46 followPath(LeftPaths::pathLeft, LeftPaths::pathRight);

47 claw.set\_value(true);

48 mogo.set\_value(true);

49 pros::delay(250);

50 lift.move\_relative(360, 200);

51 followPath(LeftPaths::pathLeftR, LeftPaths::pathRightR);

52 mogo.set\_value(false);

|  |  |  |
| --- | --- | --- |
| 53 |  | followPath(LeftPaths::fwdLeft, LeftPaths::fwdRight); |
| 54 |  |  |
| 55 |  | *//---------------------------------------------------------------------------------//* |
| 56 |  | *// AWP AUTON* |
| 57 |  | mogo.set\_value(true); |
| 58 |  | followPath(AWP::fwdRLeft, AWP::fwdRRight); |
| 59 |  | mogo.set\_value(false); |
| 60 |  | followPath(AWP::loopbackLeft, AWP::loopbackRight); |
| 61 |  | claw.set\_value(true); |
| 62 |  | leftDrive.moveRelative(-360, 600); |
| 63 |  | rightDrive.moveRelative(1500, 600); |
| 64 |  |  |
| 65 |  | *//----------------------------------------------------------------------------------* ⌋  *C→ //* |
| 66 |  | *// RIGHT AUTON* |
| 67 |  | followPath(RightPaths::curveLeft, RightPaths::curveRight); |
| 68 |  | lift.move\_relative(500, 200); |
| 69 |  | mogo.set\_value(true); |
| 70 |  | followPath(RightPaths::curveRLeft, RightPaths::curveRRight); |
| 71 |  | mogo.set\_value(false); |
| 72 |  | followPath(RightPaths::fwdLeft, RightPaths::fwdRight); |
| 73 | } |  |
| 74 |  |  |

75 void opcontrol(){

76 *// Configures brake type for drive & lift*

77 leftDrive.setBrakeMode(AbstractMotor::brakeMode::coast);

78 rightDrive.setBrakeMode(AbstractMotor::brakeMode::coast);

79 lift.set\_brake\_mode(pros::motor\_brake\_mode\_e::E\_MOTOR\_BRAKE\_BRAKE);

80

81 *// Initializes driver control variable*

82 double liftPosition = 0.0;

83 bool mogoState = false, prevBtnState = false, currentBtnState = false;

84

85 *// Initializes logo on the brain screen*

86 *// Gif gif("/usd/logo.gif", lv\_scr\_act()); // TODO - Make Gif Run in opcontrol*

87

88 while(true){

89 */\*\**

90 *\* @brief Chassis Control*

91 *\* Left Analog Y Stick -> Linear velocity the chassis drives in*

92 *\* Right Analog X Stick -> Curvature the chassis drives in*

93 *\*/*

94 double power = master.getAnalog(ControllerAnalog::leftY) \*

*C→* (abs(master.getAnalog(ControllerAnalog::leftY)) >= DEADBAND);

95 double curvature = master.getAnalog(ControllerAnalog::rightX) \*

*C→* (abs(master.getAnalog(ControllerAnalog::rightX)) >= DEADBAND);

96 auto speed = curvatureDrive(power, curvature, power == 0);

97 (chassis->getModel())->tank(speed.first, speed.second);

98

99 */\*\**

100 *\* @brief Lift Control*

101 *\* L1 (Left Top) Pressed -> Lift goes up*

102 *\* L2 (Left Bottom) Pressed -> Lift goes down*

103 *\* Both are pressed / both aren't pressed -> lift stays in the current position*

104 *\*/*

105 *// lift.moveVoltage((master.getDigital(ControllerDigital::L1) -*

*C→ master.getDigital(ControllerDigital::L2)) \* 12000);*

106 if(master.getDigital(ControllerDigital::L1)) lift.move\_voltage(12000);

107 else if(master.getDigital(ControllerDigital::L2)) lift.move\_voltage(-12000);

108 else lift.move\_voltage(0);

109

110 */\*\**

111 *\* @brief Claw Control*

112 *\* R1 (Right Top) Pressed -> claw closes*

113 *\* R1 (Right Top) not pressed -> claw opens*

114 *\*/*

115 claw.set\_value(master.getDigital(ControllerDigital::R1));

116

117 */\*\**

118 *\* @brief Mogo Holder Control*

119 *\* The solenoid toggles between the two states every time R2 (Right Bottom) is*

*C→ pressed*

120 *\*/*

121 currentBtnState = master.getDigital(ControllerDigital::R2);

122 if(currentBtnState && !prevBtnState){

!mogoState));

|  |  |
| --- | --- |
| 123 | mogo.set\_value((mogoState = |
| 124 | } |
| 125 | prevBtnState = currentBtnState; |
| 126 |  |
| 127 | pros::delay(10); |

128 }

129 }

## src/paths/awp.cpp

1 *#include "main.h"*

2

ctor<double>> AWP::fwdRLeft = {

|  |  |
| --- | --- |
| 3 std | ::vector<std::ve |
| 4 | {0}, |
| 5 | {-0.0271}, |
| 6 | {-0.0758}, |
| 7 | {-0.1254}, |
| 8 | {-0.1753}, |
| 9 | {-0.2252}, |
| 10 | {-0.2751}, |
| 11 | {-0.3251}, |
| 12 | {-0.3751}, |
| 13 | {-0.4251}, |
| 14 | {-0.475}, |
| 15 | {-0.525}, |
| 16 | {-0.575}, |
| 17 | {-0.625}, |
| 18 | {-0.6751}, |
| 19 | {-0.7251}, |
| 20 | {-0.775}, |
| 21 | {-0.825}, |
| 22 | {-0.875}, |
| 23 | {-0.925}, |
| 24 | {-0.975}, |
| 25 | {-1.025}, |
| 26 | {-1.075}, |
| 27 | {-1.125}, |
| 28 | {-1.175}, |
| 29 | {-1.225}, |
| 30 | {-1.275}, |
| 31 | {-1.325}, |
| 32 | {-1.375}, |
| 33 | {-1.425}, |
| 34 | {-1.475}, |
| 35 | {-1.525}, |
| 36 | {-1.575}, |
| 37 | {-1.625}, |
| 38 | {-1.675}, |
| 39 | {-1.725}, |
| 40 | {-1.775}, |
| 41 | {-1.825}, |
| 42 | {-1.875}, |
| 43 | {-1.9213}, |
| 44 | {-1.8979}, |
| 45 | {-1.8479}, |
| 46 | {-1.7979}, |
| 47 | {-1.7479}, |
| 48 | {-1.6979}, |
| 49 | {-1.6479}, |
| 50 | {-1.5979}, |
| 51 | {-1.5479}, |
| 52 | {-1.4979}, |

|  |  |  |
| --- | --- | --- |
| 53 |  | {-1.4479}, |
| 54 |  | {-1.3979}, |
| 55 |  | {-1.3478}, |
| 56 |  | {-1.2978}, |
| 57 |  | {-1.2478}, |
| 58 |  | {-1.1978}, |
| 59 |  | {-1.1478}, |
| 60 |  | {-1.0978}, |
| 61 |  | {-1.0478}, |
| 62 |  | {-0.9978}, |
| 63 |  | {-0.9478}, |
| 64 |  | {-0.8978}, |
| 65 |  | {-0.8478}, |
| 66 |  | {-0.7978}, |
| 67 |  | {-0.7478}, |
| 68 |  | {-0.6978}, |
| 69 |  | {-0.6477}, |
| 70 |  | {-0.5978}, |
| 71 |  | {-0.5478}, |
| 72 |  | {-0.4978}, |
| 73 |  | {-0.4477}, |
| 74 |  | {-0.3977}, |
| 75 |  | {-0.3477}, |
| 76 |  | {-0.2977}, |
| 77 |  | {-0.2476}, |
| 78 |  | {-0.1975}, |
| 79 |  | {-0.1475}, |
| 80 |  | {-0.0975}, |
| 81 |  | {-0.0473} |
| 82 | } |  |

83 ;

84 std::vector<std::vector<double>> AWP::fwdRRight = {} *//...*

85

86 std::vector<std::vector<double>> AWP::loopbackLeft = {} *//...*

87 std::vector<std::vector<double>> AWP::loopbackRight = {} *//...*

## src/paths/leftPaths.cpp

1 *#include "main.h"*

2

3 std::vector<std::vector<double>> LeftPaths::pathLeft = {} *//...*

4 std::vector<std::vector<double>> LeftPaths::pathRight = {} *//...*

5

6 std::vector<std::vector<double>> LeftPaths::pathLeftR = {} *//...*

7 std::vector<std::vector<double>> LeftPaths::pathRightR = {} *//...*

8

9 std::vector<std::vector<double>> LeftPaths::fwdLeft = {} *//...*

10 std::vector<std::vector<double>> LeftPaths::fwdRight = {} *//...*

## src/paths/rightPaths.cpp

1 *#include "main.h"*

2

3 std::vector<std::vector<double>> RightPaths::curveLeft = {} *//...*

4 std::vector<std::vector<double>> RightPaths::curveRight = {} *//...*

5

6 std::vector<std::vector<double>> RightPaths::curveRLeft = {} *//...*

7 std::vector<std::vector<double>> RightPaths::curveRRight = {} *//...*

8

9 std::vector<std::vector<double>> RightPaths::fwdLeft = {} *//...*

10 std::vector<std::vector<double>> RightPaths::fwdRight = {} *//...*