1 Header Files

1.1 include/auton.hpp

```cpp
#pragma once
#include "main.h"

/**
 * @brief Originally, our plan was to use an auton selector, but due to
 *        time constraints and bugs, we were unable to finish it on time.
 *
 */

//-------------------------------------------------------------//

/**
 * @brief Generates the trajectories used in skills autnonomous
 */
void genSkills();

/**
 * @brief Executes skills autonomous
 */
void skills();

/**
 * @brief Generates the trajectories used in red alliance's left side autnonomous
 */
void genRedLeft();

/**
 * @brief Executes red alliance's left side autonomous
 */
void redLeft();

/**
 * @brief Generates the trajectories used in red alliance's right side autnonomous
 */
void genRedRight();

/**
 * @brief Executes red alliance's right side autonomous
 */
void redRight();

/**
 * @brief Generates the trajectories used in blue alliance's left side autnonomous
 */
void genBlueLeft();

/**
 * @brief Executes blue alliance's left side autonomous
 */
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();
```

1.2 include/drive.hpp

```cpp
#pragma once
#include "main.h"

/**
 * @brief Computes the desired drive speed to control the chassis using curvature control
 *        The turn power is scaled with the linear power in order to allow finer control
 *        of the chassis.
 *
 * @param moveC - the desired linear speed for the robot to move in, normalized to [-1, 1]
 * @param turnC - the desired curvature for the robot to move in, normalized to [-1, 1]
 * @param quickTurn - when this parameter is set to true, arcade control is used instead to allow point turn
 *
 * @return the desired left and right velocity for the chassis motors to move in, normalized to [-1, 1]
 */
std::pair<double, double> curvatureDrive(double moveC, double turnC, bool quickTurn);

/**
 * @brief Turns the robot to the desired global angle (using closed-loop control)
 *
 * @param targetAngle - the target odometry global angle to turn to, normalized to [-2pi, 2pi]
 */
void turnToAngle(okapi::QAngle targetAngle);

/**
 * @brief This is our custom velocity controller. Although the motor has decent internal
 *        PID velocity control, through our testing, we realized that it still is not
 *        as precise as this custom velocity controller. Yet due to time constraints,
 *        we did not have enough to time to individually tune our motors, and thus, we
 *        stuck with using the motor's internal velocity controller.
 *
 *        The reason why we even have this is because our drive's movement in auton heavily
 *        relies on 2D motion profiles. There is lots of complicated math behind it, but in
 *        a sense, we create a smooth trajectory for our robot to follow and convert it to
 *        actual drive velocity that our robot follows for every 10 ms. As a result, the
 *        how precise our auton is heavily relies on how well the velocity control of our
 *        motors are.
 *
 * @param velocity desired velocity
 * @param accel desired acceleration
 * @param currSpeed current speed of the motor(s)
 * @return output for the motor(s)
 */
double velControl(QSpeed velocity, QAcceleration accel, QSpeed currSpeed);

```

```
45  /**
46   * @brief The trajectory generator we use outputs linear velocity (ft/s). In order for
    ↪  us to
47   *        make use of that, we must convert the linear velocity to RPM, which the motors
    ↪  follow.
48   *        Although there are much more mathematical ways of making this conversion (such
    ↪  as
49   *        recording the wheel size, gear ratio, and motor speed, and using those values
    ↪  to
50   *        convert) we decided to use a much simpler method of calculating ratios. Since
    ↪  the
51   *        maximum motor RPM (for our drive) is 600 RPM and the maximum linear velocity
    ↪  is
52   *        4.92126 ft/s, we can use simple ratios to convert the desired linear velocity
    ↪  to RPM.
53   *
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
    ↪  inputs
61   *        of the desired left and right velocites and feeds it to the drive motor to
    ↪  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,
    ↪  std::vector<std::vector<double>> rightVel);
```

1.3 include/globals.hpp

```cpp
#pragma once
#include "main.h"

/**
 * @brief set 'Pneumatic' as pros::ADIDigitalOut
 *
 */
using Pneumatic = pros::ADIDigitalOut;

// CONSTANTS
/**
 * @brief Here is our constants (though we only ended up really using
 *        DEADBAND since we had no time to tune lift)
 *
 */
const double DEADBAND = 0.0500;
const double MAXLIFTHEIGHT = 2000;
const double LIFT_INCREMENT = 10;

// CONTROLLER
/**
 * @brief Our controller
 *
 */
extern Controller master;

// MOTORS
/**
 * @brief These are all the motors we used
 *        (each of our drive motor groups have 3 individual motors)
 *
 */
extern MotorGroup leftDrive;
extern MotorGroup rightDrive;
extern pros::Motor lift;
extern Motor roller;

// SENSORS
/**
 * @brief These were all sensors we were planning on using. But due to
 *        time constraints, mechanical issues, and our insanely accurate
 *        pathing algorithm, we found no need to use these sensors.
 *
 */
extern ADIEncoder trackLeft;
extern ADIEncoder trackRight;
extern ADIEncoder trackMiddle;
extern RotationSensor liftSensor;
extern IMU imu;

// PNEUMATICS
/**
```

```
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;
```

1.4 include/main.h

```
/**
 * \file main.h
 *
 * Contains common definitions and header files used throughout your PROS
 * project.
 *
 * Copyright (c) 2017-2021, Purdue University ACM SIGBots.
 * All rights reserved.
 *
 * This Source Code Form is subject to the terms of the Mozilla Public
 * License, v. 2.0. If a copy of the MPL was not distributed with this
 * file, You can obtain one at http://mozilla.org/MPL/2.0/.
 */

#ifndef _PROS_MAIN_H_
#define _PROS_MAIN_H_

/**
 * If defined, some commonly used enums will have preprocessor macros which give
 * a shorter, more convenient naming pattern. If this isn't desired, simply
 * comment the following line out.
 *
 * For instance, E_CONTROLLER_MASTER has a shorter name: CONTROLLER_MASTER.
 * E_CONTROLLER_MASTER is pedantically correct within the PROS styleguide, but
 * not convienent for most student programmers.
 */
#define PROS_USE_SIMPLE_NAMES

/**
 * If defined, C++ literals will be available for use. All literals are in the
 * pros::literals namespace.
 *
 * For instance, you can do `4_mtr = 50` to set motor 4's target velocity to 50
 */
#define PROS_USE_LITERALS

#include "api.h"

/**
 * You should add more #includes here
 */
#include "okapi/api.hpp"
#include "pros/api_legacy.h"
#include "pros/apix.h"

/**
 * If you find doing pros::Motor() to be tedious and you'd prefer just to do
 * Motor, you can use the namespace with the following commented out line.
 *
 * IMPORTANT: Only the okapi or pros namespace may be used, not both
 * concurrently! The okapi namespace will export all symbols inside the pros
 * namespace.
```

```
 */
// using namespace pros;
// using namespace pros::literals;
using namespace okapi;

/**
 * Prototypes for the competition control tasks are redefined here to ensure
 * that they can be called from user code (i.e. calling autonomous from a
 * button press in opcontrol() for testing purposes).
 */
#ifdef __cplusplus
extern "C" {
#endif
void autonomous(void);
void initialize(void);
void disabled(void);
void competition_initialize(void);
void opcontrol(void);
#ifdef __cplusplus
}
#endif

#ifdef __cplusplus
/**
 * You can add C++-only headers here
 */
#include <iostream>
#include <cmath>
#include <algorithm>
#include <map>
#include "globals.hpp"
#include "drive.hpp"
#include "auton.hpp"
#include "paths/leftPaths.hpp"
#include "paths/awp.hpp"
#include "paths/rightPaths.hpp"
#include "gif-pros/gifclass.hpp"


#endif

#endif  // _PROS_MAIN_H_
```

1.5 include/gif-pros/gifclass.hpp

```cpp
#pragma once
#include "main.h"
#include "gifdec.h"

/**
 * MIT License
 * Copyright (c) 2019 Theo Lemay
 * https://github.com/theol0403/gif-pros
 */

class Gif {

public:

  /**
   * Construct the Gif class
   * @param fname  the gif filename on the SD card (prefixed with /usd/)
   * @param parent the LVGL parent object
   */
  Gif(const char* fname, lv_obj_t* parent);

  /**
   * Destructs and cleans the Gif class
   */
  ~Gif();

  /**
   * Pauses the GIF task
   */
  void pause();

  /**
   * Resumes the GIF task
   */
  void resume();

  /**
   * Deletes GIF and frees all allocated memory
   */
  void clean();

private:

  gd_GIF* _gif = nullptr; // gif decoder object
  void* _gifmem = nullptr; // gif file loaded from SD into memory
  uint8_t* _buffer = nullptr; // decoder frame buffer

  lv_color_t* _cbuf = nullptr; // canvas buffer
  lv_obj_t* _canvas = nullptr; // canvas object

  pros::task_t _task = nullptr; // render task

```

```
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  };
```

1.6 include/gif-pros/gifdec.h

```c
#ifndef GIFDEC_H
#define GIFDEC_H

#include <stdio.h>
#include <stdint.h>
#include <sys/types.h>

#ifdef __cplusplus
extern "C" {
#endif

#define BYTES_PER_PIXEL 4

typedef struct gd_Palette {
    int size;
    uint8_t colors[0x100 * 3];
} gd_Palette;

typedef struct gd_GCE {
    uint16_t delay;
    uint8_t tindex;
    uint8_t disposal;
    int input;
    int transparency;
} gd_GCE;

typedef struct gd_GIF {
    FILE *fp;
    off_t anim_start;
    uint16_t width, height;
    uint16_t depth;
    uint16_t loop_count;
    gd_GCE gce;
    gd_Palette *palette;
    gd_Palette lct, gct;
    void (*plain_text)(
        struct gd_GIF *gif, uint16_t tx, uint16_t ty,
        uint16_t tw, uint16_t th, uint8_t cw, uint8_t ch,
        uint8_t fg, uint8_t bg
    );
    void (*comment)(struct gd_GIF *gif);
    void (*application)(struct gd_GIF *gif, char id[8], char auth[3]);
    uint16_t fx, fy, fw, fh;
    uint8_t bgindex;
    uint8_t *canvas, *frame;
} gd_GIF;

gd_GIF *gd_open_gif(FILE *fp);
int gd_get_frame(gd_GIF *gif);
void gd_render_frame(gd_GIF *gif, uint8_t *buffer);
void gd_rewind(gd_GIF *gif);
void gd_close_gif(gd_GIF *gif);
```

```
53
54  #ifdef __cplusplus
55  }
56  #endif
57
58  #endif /* GIFDEC_H */
```

1.7 include/paths/awp.hpp

```
#pragma once
#include "main.h"

/**
 * @brief The follow class contains all paths used in the auton win point (AWP) auton
 *
 */
class AWP {
    public:
    static std::vector<std::vector<double>> fwdRLeft;
    static std::vector<std::vector<double>> fwdRRight;

    static std::vector<std::vector<double>> loopbackLeft;
    static std::vector<std::vector<double>> loopbackRight;
};
```

1.8 include/paths/leftPaths.hpp

```hpp
#pragma once
#include "main.h"

/**
 * @brief The following class contains all paths used in the left auton.
 *        In the auton, we first capture the left neutral mogo, score a ring
 *        and bring it back. As we head back to our home row, we move to our
 *        alliance mogo where we deposit a ring into the base of our mogo.
 *
 */
class LeftPaths {
    public:
    static std::vector<std::vector<double>> pathLeft;
    static std::vector<std::vector<double>> pathRight;

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

    static std::vector<std::vector<double>> fwdLeft;
    static std::vector<std::vector<double>> fwdRight;
};
```

1.9 include/paths/rightPaths.hpp

```hpp
#pragma once
#include "main.h"

/**
 * @brief The following class contains all paths used in the right auton.
 *        The right auton begins by curving to the tall neutral mogo, tipping
 *        it over, and going back to score 2 rings on our right alliance mogo.
 *
 */
class RightPaths {
    public:
    static std::vector<std::vector<double>> curveLeft;
    static std::vector<std::vector<double>> curveRight;

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

    static std::vector<std::vector<double>> fwdLeft;
    static std::vector<std::vector<double>> fwdRight;
};
```

2 Source Files

2.1 src/auton.cpp

```
/**
 * @brief Unfortunately due to time constraints and bugs, we were never
 *        able to finish our auton selector/organizational system
 *
 */

// #include "main.h"

// void genSkills(){}

// void skills(){
//     chassis->setState({0_in, 0_in, 0_deg});
// }

// void genRedLeft(){
//     profiler->generatePath({{0_in, 0_in, 0_deg}, {0_in, 24_in, 0_deg}}, "Test");
// }

// void redLeft(){
//     chassis->setState({0_in, 0_in, 0_deg});
//     profiler->setTarget("Test");
//     profiler->waitUntilSettled();

//     turnToAngle(90_deg);
// }

// void genRedRight(){}

// void redRight(){
//     chassis->setState({0_in, 0_in, 0_deg});
// }

// void genBlueLeft(){}

// void blueLeft(){
//     chassis->setState({0_in, 0_in, 0_deg});
// }

// void genBlueRight(){}

// void blueRight(){
//     chassis->setState({0_in, 0_in, 0_deg});
// }

// void genAwp(){}

// void awp(){
//     chassis->setState({0_in, 0_in, 0_deg});
// }
```

2.2 src/drive.cpp

```cpp
#include "main.h"

std::pair<double, double> curvatureDrive(double moveC, double turnC, bool quickTurn){
    // Compute velocity, right stick = curvature if no quickturn, else power
    double leftSpeed = moveC + (quickTurn ? turnC : abs(moveC) * turnC);
    double rightSpeed = moveC - (quickTurn ? turnC : abs(moveC) * turnC);

    // Normalize velocity
    double maxMagnitude = std::max(abs(leftSpeed), abs(rightSpeed));
    if (maxMagnitude > 1.0) {
        leftSpeed /= maxMagnitude;
        rightSpeed /= maxMagnitude;
    }

    return std::make_pair(leftSpeed, rightSpeed);
}

double velControl(QSpeed velocity, QAcceleration accel, QSpeed currSpeed) {
    double kV = 0.0, kA = 0.0, kP = 0.0;
    return kV * velocity.convert(okapi::mps) + kA * accel.convert(okapi::mps2) + kP *
    ↪  (velocity - currSpeed).convert(okapi::mps);
}

// velocity only, doesn't use custon velControl
std::vector<double> pathToRPM(std::vector<std::vector<double>> path) {
    std::vector<double> newPath;
    double val = 0.0;
    for(int i = 0; i < path.size(); i++) {
        // max vel = 4.92126 ft/s, max rpm = 600 --> convert values to rpm
        val = (path[i][0] * 600) / 4.92126;
        newPath.push_back(val);
    }
    return newPath;
}

void followPath(std::vector<std::vector<double>> leftVel,
↪  std::vector<std::vector<double>> rightVel) {
    std::vector<double> left = pathToRPM(leftVel); std::vector<double> right =
    ↪  pathToRPM(rightVel);
    for(int i = 0; i < left.size() || i < right.size(); i++) {
        leftDrive.moveVelocity(left[i]);
        rightDrive.moveVelocity(right[i]);
        pros::delay(10);
        // pros::Task::delay_until(&now, 10);
    }
    leftDrive.moveVelocity(0);
    rightDrive.moveVelocity(0);
}

// void turnToAngle(okapi::QAngle targetAngle){
//   turnPID->reset();
//   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  // }
```

2.3 src/globals.cpp

```cpp
#include "main.h"

// CONTROLLER
Controller master(ControllerId::master);

// MOTORS
Motor leftTop(4, false, AbstractMotor::gearset::blue, AbstractMotor::encoderUnits::degrees);
Motor leftMiddle(5, true, AbstractMotor::gearset::blue, AbstractMotor::encoderUnits::degrees);
Motor leftBottom(6, false, AbstractMotor::gearset::blue, AbstractMotor::encoderUnits::degrees);
Motor rightTop(7, true, AbstractMotor::gearset::blue, AbstractMotor::encoderUnits::degrees);
Motor rightMiddle(8, false, AbstractMotor::gearset::blue, AbstractMotor::encoderUnits::degrees);
Motor rightBottom(9, true, AbstractMotor::gearset::blue, AbstractMotor::encoderUnits::degrees);
MotorGroup leftDrive({leftTop, leftMiddle, leftBottom});
MotorGroup rightDrive({rightTop, rightMiddle, rightBottom});
// Motor lift(10, true, AbstractMotor::gearset::green, AbstractMotor::encoderUnits::degrees);
pros::Motor lift(10, true);
// Motor roller(8, false, AbstractMotor::gearset::blue, AbstractMotor::encoderUnits::degrees); // TODO - Change Port

// SENSORS
ADIEncoder trackLeft({14, 'A', 'B'}, false); // TODO - Change Port, reverse?
ADIEncoder trackRight({14, 'C', 'D'}, false); // TODO - Change Port, reverse?
ADIEncoder trackMiddle({14, 'E', 'F'}, false); // TODO - Change Port, reverse?
RotationSensor liftSensor(20, false); // TODO - Change Port, reverse?
IMU imu(14);

// PNEUMATICS
Pneumatic mogo('C');
Pneumatic claw('D'); // TODO - Change Port

// SUBSYSTEM CONTROLLERS
std::shared_ptr<ChassisController> chassis = ChassisControllerBuilder()
  .withMotors(leftDrive, rightDrive)
  .withDimensions({AbstractMotor::gearset::blue, 5.0/3.0}, {{3.25_in, 38.5_cm}, imev5BlueTPR}) // TODO - Change Track Width
  // .withSensors(trackLeft, trackRight, trackMiddle)
  // .withOdometry({{2.75_in, 6.25_in}, quadEncoderTPR}) // TODO - Change Track Width
    // .withOdometry()
  // .buildOdometry();
    .build();

std::shared_ptr<AsyncMotionProfileController> profiler =
    AsyncMotionProfileControllerBuilder()
    .withLimits({ // TODO - Tune Max Robot Velocity / Acceleration
        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 =
    ↪  AsyncPosControllerBuilder()
50  //      .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 =
    ↪  std::make_shared<IterativePosPIDController>(0, 0, 0, 0,
    ↪  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;
```

2.4 src/main.cpp

```cpp
#include "main.h"

void initialize(){
  pros::lcd::initialize();
    master.setText(0, 0, "Current Autonomous: " + std::to_string(selectedAuton));
    pros::lcd::set_text(4, "init");


    // Initializes Controller
    // liftController->tarePosition();

    // Adds autonomous
    // auton.insert(std::make_pair(0, [](){})); // lambda ftw
    // auton.insert(std::make_pair(1, redLeft));
    // auton.insert(std::make_pair(2, redRight));
    // auton.insert(std::make_pair(3, blueLeft));
    // auton.insert(std::make_pair(4, blueRight));
    // auton.insert(std::make_pair(5, awp));
    // auton.insert(std::make_pair(6, skills));

    // // Adds path generation functions
    // path.insert(std::make_pair(0, [](){}));
    // path.insert(std::make_pair(1, genRedLeft));
    // path.insert(std::make_pair(2, genRedRight));
    // path.insert(std::make_pair(3, genBlueLeft));
    // path.insert(std::make_pair(4, genBlueRight));
    // path.insert(std::make_pair(5, genAwp));
    // path.insert(std::make_pair(6, genSkills));

    // // Generates path based on pre-selected auton
    // path[selectedAuton]();
}

void disabled(){}

void competition_initialize(){}

void autonomous(){
    // INITIALIZATION
    lift.set_brake_mode(pros::motor_brake_mode_e::E_MOTOR_BRAKE_HOLD);
    leftDrive.setBrakeMode(AbstractMotor::brakeMode::hold);
    rightDrive.setBrakeMode(AbstractMotor::brakeMode::hold);

    //-------------------------------------------------------------------------//
    // LEFT AUTON
    followPath(LeftPaths::pathLeft, LeftPaths::pathRight);
    claw.set_value(true);
    mogo.set_value(true);
    pros::delay(250);
    lift.move_relative(360, 200);
    followPath(LeftPaths::pathLeftR, LeftPaths::pathRightR);
    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      //-------------------------------------------------------------------------------
        ↪  -------//
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) *
          ↪  (abs(master.getAnalog(ControllerAnalog::leftY)) >= DEADBAND);
95          double curvature = master.getAnalog(ControllerAnalog::rightX) *
            ↪  (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) -
                ↪  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
    ↪  pressed
120              */
121             currentBtnState = master.getDigital(ControllerDigital::R2);
122             if(currentBtnState && !prevBtnState){
123                 mogo.set_value((mogoState = !mogoState));
124             }
125             prevBtnState = currentBtnState;
126 
127             pros::delay(10);
128         }
129     }
```

2.5 src/paths/awp.cpp

```cpp
#include "main.h"

std::vector<std::vector<double>> AWP::fwdRLeft = {
        {0},
        {-0.0271},
        {-0.0758},
        {-0.1254},
        {-0.1753},
        {-0.2252},
        {-0.2751},
        {-0.3251},
        {-0.3751},
        {-0.4251},
        {-0.475},
        {-0.525},
        {-0.575},
        {-0.625},
        {-0.6751},
        {-0.7251},
        {-0.775},
        {-0.825},
        {-0.875},
        {-0.925},
        {-0.975},
        {-1.025},
        {-1.075},
        {-1.125},
        {-1.175},
        {-1.225},
        {-1.275},
        {-1.325},
        {-1.375},
        {-1.425},
        {-1.475},
        {-1.525},
        {-1.575},
        {-1.625},
        {-1.675},
        {-1.725},
        {-1.775},
        {-1.825},
        {-1.875},
        {-1.9213},
        {-1.8979},
        {-1.8479},
        {-1.7979},
        {-1.7479},
        {-1.6979},
        {-1.6479},
        {-1.5979},
        {-1.5479},
        {-1.4979},
```

```
            {-1.4479},
            {-1.3979},
            {-1.3478},
            {-1.2978},
            {-1.2478},
            {-1.1978},
            {-1.1478},
            {-1.0978},
            {-1.0478},
            {-0.9978},
            {-0.9478},
            {-0.8978},
            {-0.8478},
            {-0.7978},
            {-0.7478},
            {-0.6978},
            {-0.6477},
            {-0.5978},
            {-0.5478},
            {-0.4978},
            {-0.4477},
            {-0.3977},
            {-0.3477},
            {-0.2977},
            {-0.2476},
            {-0.1975},
            {-0.1475},
            {-0.0975},
            {-0.0473}
        }
;
std::vector<std::vector<double>> AWP::fwdRRight = {} //...

std::vector<std::vector<double>> AWP::loopbackLeft = {} //...
std::vector<std::vector<double>> AWP::loopbackRight = {} //...
```

2.6 src/paths/leftPaths.cpp

```cpp
#include "main.h"

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

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

std::vector<std::vector<double>> LeftPaths::fwdLeft = {} //...
std::vector<std::vector<double>> LeftPaths::fwdRight = {} //...
```

2.7 src/paths/rightPaths.cpp

```cpp
#include "main.h"

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

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

std::vector<std::vector<double>> RightPaths::fwdLeft = {} //...
std::vector<std::vector<double>> RightPaths::fwdRight = {} //...
```