

1 README.md

2 Header Files

2.1 include/Auton.hpp

```
1  /**
2   * Auton.hpp
3   *
4   * This contains the declaration of the Auton struct,
5   * which is responsible for reading the sd card to determine
6   * which auton is selected, and running the correct auton.
7   */
8  #pragma once // makes sure the file is only included once
9  #include "main.h" // gives access to dependancies from other files
10 class Auton
11 {
12     public:
13         // when making autons, you must add the text to the dropdown in DisplayControl.cpp,
14         ↪ a new value
15         // to this enum, and a new case is the switch in Auton.cpp
16         enum class Autons // all possible autons
17         {
18             none,
19             test,
20             prog
21         } static auton; // selected auton
22
23         static void readSettings(); // read the sd card to set the settings
24
25         static void runAuton(); // runs the selected auton
26
27     private:
28         static void auton_task_func(void *); // separate thread for running the auton, in
29         ↪ case a
30
31                                     // particular auton needs control over it's
32                                     ↪ thread.
33 };
```

2.2 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) 2017-2020, 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/.
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 convenient 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/apix.h"
44  //#include "pros/api_legacy.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  #include "movement/paths/ProfileStep.hpp"
59  #include "movement/paths/Trajectory.hpp"
60
61  #include "gui/odomDebug.hpp"
62  #include "util/CustomOdometry.hpp"
63  #include "gui/DisplayControl.hpp"
64  #include "util/util.hpp"
65  #include "util/definitions.hpp"
66
67  #include "movement/paths/SimplePath.hpp"
68
69  #include "movement/AsyncAction.hpp"
70  #include "movement/Drivetrain.hpp"
71
72  #include "stateMachines/VStateMachine.hpp"
73  #include "stateMachines/DrivetrainStateMachine.hpp"
74  #include "stateMachines/BallControlStateMachine.hpp"
75
76  #include "Auton.hpp"
77
78  /**
79   * Prototypes for the competition control tasks are redefined here to ensure
80   * that they can be called from user code (i.e. calling autonomous from a
81   * button press in opcontrol() for testing purposes).
82   */
83  #ifdef __cplusplus
84  extern "C"
85  {
86  #endif
87      void autonomous(void);
88      void initialize(void);
89      void disabled(void);
90      void competition_initialize(void);
91      void opcontrol(void);
92  #ifdef __cplusplus
93  }
94  #endif
95
96  #ifdef __cplusplus
97  /**
98   * You can add C++-only headers here
99   */
100  // #include <iostream>
101  #endif
102
103  #endif // _PROS_MAIN_H_

```

2.3 include/gui/DisplayControl.hpp

```
1  /**
2   * DisplayControl.hpp
3   *
4   * This file contains the declaration of the DisplayControl class.
5   * DisplayControl is the class that handles the organization of the
6   * LittleV Graphics Library (LVGL) objects on the screen of the brain.
7   */
8  #pragma once // makes sure the file is only included once
9  #include "main.h" // gives access to objects not declared here (LVGL objects)
10 class DisplayControl
11 {
12
13     /* ----- Tabview Elements ----- */
14     static lv_obj_t * mtabview; // contains the whole tabview
15
16     static lv_obj_t * mtabview_odom; // tabview page with odom debugger
17     OdomDebug modom; // odom debugger
18
19     static lv_obj_t * mtabview_auton; // tab for setting auton
20     static lv_obj_t * mtabview_auton_dropdown; // autons to choose from
21     static lv_res_t tabview_auton_dropdown_action(lv_obj_t * idropdown); // event
22     ↪ handler
23
24     static lv_obj_t * mtabview_graph; // tabview page with graph
25     static lv_obj_t * mtabview_graph_chart; // graph
26     static lv_chart_series_t * mtabview_graph_chart_series_0; // chart series...
27     static lv_chart_series_t * mtabview_graph_chart_series_1;
28     static lv_chart_series_t * mtabview_graph_chart_series_2;
29     static lv_chart_series_t * mtabview_graph_chart_series_3;
30     static lv_chart_series_t * mtabview_graph_chart_series_4;
31     static lv_chart_series_t * mtabview_graph_chart_series_5;
32     static lv_chart_series_t * mtabview_graph_chart_series_6;
33
34     static lv_obj_t * mtabview_misc; // extra tabview page for anything
35     static lv_obj_t * mtabview_misc_container; // container on the misc page to hold
36     ↪ elements
37     static lv_obj_t * mtabview_misc_label; // text box on misc page
38     static lv_obj_t * mtabview_misc_label_2; // second text box on misc page
39
40     /* ----- Styles ----- */
41     static lv_style_t mstyle_tabview_indic; // for page indicator line
42     static lv_style_t mstyle_tabview_btn; // for page header
43     static lv_style_t mstyle_tabview_btn_tgl; // for selected page header
44     static lv_style_t mstyle_tabview_btn_pr; // for pressed page header
45     static lv_style_t mstyle_tabview_container; // for page background
46     static lv_style_t mstyle_text; // for text
47
48     public:
49     DisplayControl(); // constructor that sets everything up, like styles and
50     ↪ positioning
51
52     void setOdomData(); // updates the values for OdomDebug
```

```

50 void setAutonDropdown(); // updates the dropdown to match sd card at the start of
   ↳ the program,
51                               // to ensure the auton set on the sd card is always the
   ↳ same as the
52                               // auton displayed on the dropdown.
53
54 void setChartData(int iseries,
55                  double ivalue); // sets the value of one of the series in the
   ↳ chart
56
57 void setMiscData(int ilabel,
58                 std::string itext); // sets the information displayed in the misc
   ↳ tab
59 };
60
61 namespace def
62 {
63 extern DisplayControl
64 display; // declares the display object as extern, to make sure it only gets
   ↳ constructed once
65 }

```

2.4 include/gui/odomDebug.hpp

```
1  /**
2   * odomDebug.hpp
3   *
4   * The contents of this file were not written by any members of 333A*.
5   * This is code from the publicly available GitHub repository, odomDebug
6   * by theol0403, found here: https://github.com/theol0403/odomDebug.
7   *
8   * The OdomDebug class is used for the tab on the screen of the brain
9   * that shows the odometry position of the robot in the form of number
10  * values and a moving circle on a picture of the field representing the
11  * robot.
12  *
13  *   * slight modifications were made to make it work with the display
14  */
15 #pragma once
16 #include "main.h"
17
18 class OdomDebug
19 {
20
21 public:
22     /**
23      * Contains robot state - x, y, theta
24      * Can be initialized using QUnits or doubles
25      */
26     struct state_t
27     {
28         QLength x{0.0};
29         QLength y{0.0};
30         QAngle theta{0.0};
31
32         /**
33          * @param ix QLength
34          * @param iy QLength
35          * @param itheta QAngle
36          */
37         state_t(QLength ix, QLength iy, QAngle itheta);
38
39         /**
40          * @param ix inches
41          * @param iy inches
42          * @param itheta radians
43          */
44         state_t(double ix, double iy, double itheta);
45     };
46
47     /**
48      * Contains encoder information - left, right, middle(optional)
49      * Can be initialized using three or two sensors
50      */
51     struct sensors_t
52     {
```

```

53     double left{0.0};
54     double right{0.0};
55     double middle{0.0};
56
57     /**
58      * @param ileft the left encoder value
59      * @param iright the right encoder value
60      */
61     sensors_t(double ileft, double iright);
62
63     /**
64      * @param ileft the left encoder value
65      * @param iright the right encoder value
66      * @param imiddle imiddle the middle encoder value
67      */
68     sensors_t(double ileft, double iright, double imiddle);
69
70     private:
71     bool hasMiddle{false};
72     friend class OdomDebug;
73 };
74
75 /**
76  * Constructs the OdomDebug object.
77  * @param parent the lvgl parent, color is inherited
78  */
79 OdomDebug(lv_obj_t * parent);
80
81 /**
82  * Constructs the OdomDebug object.
83  * @param parent the lvgl parent
84  * @param mainColor The main color for the display
85  */
86 OdomDebug(lv_obj_t * parent, lv_color_t mainColor);
87
88 ~OdomDebug();
89
90 /**
91  * Sets the function to be called when a tile is pressed
92  * @param callback a function that sets the odometry state
93  */
94 void setStateCallback(std::function<void(state_t state)> callback);
95
96 /**
97  * Sets the function to be called when the reset button is pressed
98  * @param callback a function that resets the odometry and sensors
99  */
100 void setResetCallback(std::function<void()> callback);
101
102 /**
103  * Sets the position of the robot in QUnits and puts the sensor data on the
104  * display
105  * @param state robot state - x, y, theta
106  * @param sensors encoder information - left, right, middle (optional)

```

```

107     */
108     void setData(state_t state, sensors_t sensors);
109
110 private:
111     // parent container
112     lv_obj_t * container = nullptr;
113     lv_style_t cStyle;
114
115     // field
116     lv_style_t fStyle;
117     double fieldDim = 0; // width and height of field container
118
119     // tile styles
120     lv_style_t grey;
121     lv_style_t red;
122     lv_style_t blue;
123
124     // robot point
125     lv_obj_t * led = nullptr;
126     lv_style_t ledStyle;
127
128     // robot line
129     lv_obj_t * line = nullptr;
130     lv_style_t lineStyle;
131     std::vector<lv_point_t> linePoints = {{0, 0}, {0, 0}}; // line positions
132     int lineWidth = 0;
133     int lineLength = 0;
134
135     // status label
136     lv_obj_t * statusLabel = nullptr;
137     lv_style_t textStyle;
138
139     // reset button styles
140     lv_style_t resetRel;
141     lv_style_t resetPr;
142
143     // external callbacks to interface with odometry
144     std::function<void(state_t state)> stateFnc = nullptr;
145     std::function<void()> resetFnc = nullptr;
146
147     static lv_res_t tileAction(lv_obj_t *); // action when tile is pressed
148     static lv_res_t resetAction(lv_obj_t *); // action when reset button is pressed
149 };

```


2.5 include/movement/AsyncAction.hpp

```
1  /**
2   * AsyncAction.hpp
3   *
4   * This file contains the definition of the AsyncAction struct.
5   * AsyncActions are objects that have an action (maction) and
6   * a certain error where the action should be executed (merror).
7   * It is used by motions in the Drivetrain class to run asynchronous
8   * actions at a certain distance from the target.
9   */
10 #pragma once // makes sure the file is only included once
11 #include "main.h" // gives access to objects declared elsewhere
12 struct AsyncAction
13 {
14     AsyncAction(double ierror, std::function<void()> iaction)
15         : merror(ierror), maction(iaction) // constructor
16     {
17     }
18
19     double merror; // error value at which the loop will execute the action
20     std::function<void()> maction; // action to execute
21 };
```

2.6 include/movement/Drivetrain.hpp

```
1  /**
2   * Drivetrain.hpp
3   *
4   * This file contains the declaration of the Drivetrain class.
5   * The Drivetrain class handles almost everything relating to the
6   * drivetrain: motor control, settings (like max speed), basic
7   * movement methods (like tank or arcade), more advanced movement
8   * methods (like PID to point, path following, and motion
9   * profiling), and more.
10  */
11  #pragma once // makes sure the file is only included once
12  #include "main.h" // gives access to dependencies from other files
13  class Drivetrain // creates the class for the drivetrain
14  {
15  private:
16      /* ----- Motor References ----- */
17      static Motor & mmtrLeftFront;
18      static Motor & mmtrRightFront;
19      static Motor & mmtrRightBack;
20      static Motor & mmtrLeftBack;
21
22      /* ----- Chassis ----- */
23      static std::shared_ptr<ChassisController>
24          mchassis; // chassis object for using Pathfinder through okapi
25
26  protected:
27      /* ----- Settings ----- */
28      static double mmaxSpeed;
29      static bool menabled;
30
31      /* ----- SimpleFollow Data ----- */
32      static double mlastLookIndex; // index of the last lookahead point
33      static double
34          mlastPartialIndex; // fractional index of where the last lookahead point was on
35                             ↪ the segment
36
37      /* ----- Odometry Accessors ----- */
38      static OdomState getState(); // get position as OdomState
39      static QLength getXPos();
40      static QLength getYPos();
41      static QAngle getTheta();
42      static ExtendedPoint getPoint(); // get position as ExtendedPoint
43
44      /* ----- Helpers ----- */
45      static QAngle
46          angleToPoint(const Point & itargetPoint); // calculates the field centric direction
47                                                     ↪ to the
48                                                     // itargetPoint from the robot's current
49                                                     ↪ position
50      static std::optional<double> findIntersection(
51          ExtendedPoint istart, ExtendedPoint iend,
```

```

49     const double & ilookDistIn); // looks for interections between the line segment
    ↪ created by
50                                     // the two points (istart and iend), and the
    ↪ circle around the
51                                     // robot with radius ilookDistIn (lookahead circle)
52 static ExtendedPoint
53 findLookahead(SimplePath ipath,
54               const double & mlookDistIn); // looks for the intersection point
    ↪ between the
55                                     // lookahead circle and the SimplePath,
    ↪ ipath
56
57 public:
58 /* ----- Getters/Setters ----- */
59 static double getMaxSpeed();
60 static void setMaxSpeed(const double imaxSpeed);
61
62 static bool isEnabled();
63 static void enable(); // allows movements to be startable
64 static void disable(); // stops active movements
65
66 static void
67 checkNextAsync(const double & ierror,
68               std::vector<AsyncAction> &
69               iactions); // checks if the next AsyncAction should execute, and
    ↪ executes it
70                           // (and removes it from the list) if it should
71
72 /* ----- Basic Movement ----- */
73 static void moveIndependant(
74     double ileftFront, double  irightFront, double  irightBack, double  ileftBack,
75     const bool idesaturate = true); // moves each motor {lf, rf, rb, lb} in range
    ↪ [-1,1]
76 static void
77 moveTank(const double ileft, const double  iright,
78         const bool idesaturate =
79         true); // spins the left side and right side motors at certian speeds
    ↪ [-1,1]
80 static void moveArcade(
81     const double iforward, const double  istrafe, const double  iturn,
82     const bool idesaturate = true); // moves the robot with arcade-style inputs
    ↪ (range[-1,1])
83
84 /* ----- Intermediate Movement ----- */
85 static void moveInDirection(
86     QAngle idirection, const bool ifieldCentric = false, double  imagnitude = 1,
87     double  itheta = 0,
88     const bool idesaturate = true); // moves the robot with a certain speed in a
    ↪ certain
89                                     // direction, while turning a certain amount
90
91 /* ----- Move to Point Methods ----- */
92 static void strafeToPoint(
93     ExtendedPoint iPoint, std::vector<AsyncAction> iactions = {},

```

```

94     PID imagitudePID = PID(0.4, 0.005, 2.6, 0.5, 0.25, 0.05, 10_ms),
95     PID iturnPID = PID(0.028, 0.0, 0.08, 0.0, 1.5, 0.1, 10_ms),
96     Slew imagitudeSlew = Slew(1000, 1000),
97     Slew iturnSlew =
98         Slew(1000, 1000)); // drives in a stright line to the point while turning
99         ↪ using set
100
101                                     // PID/Slew gains, and executing the AsyncActions at the
102                                     ↪ right times
103
104 static void straightToPoint(
105     ExtendedPoint itarget, std::vector<AsyncAction> iactions = {}, QLength
106     ↪ inoTurnRange = 3_in,
107     double iturnWeight = 1.7, PID imagitudePID = PID(0.4, 0.005, 2.6, 0.5, 0.25,
108     ↪ 0.05, 10_ms),
109     PID iturnPID = PID(0.028, 0.0, 0.08, 0.0, 1.5, 0.1, 10_ms),
110     Slew imagitudeSlew = Slew(1000, 1000),
111     Slew iturnSlew = Slew(1000,
112     ↪ 1000)); // drives to the point without strafing using set
113     ↪ PID/Slew
114     // gains, and executing the AsyncActions at the
115     ↪ right times
116
117 static void arcStraightToPoint(
118     ExtendedPoint itarget, std::vector<AsyncAction> iactions = {}, double
119     ↪ iweightModifier = 10,
120     QLength inoTurnRange = 3_in,
121     PID imagitudePID = PID(0.4, 0.005, 2.6, 0.5, 0.25, 0.05, 10_ms),
122     PID iturnPID = PID(0.028, 0.0, 0.08, 0.0, 1.5, 0.1, 10_ms),
123     Slew imagitudeSlew = Slew(1000, 1000),
124     Slew iturnSlew = Slew(
125     ↪ 1000,
126     ↪ 1000)); // drive in an "arc" (doesn't follow a path, just approximates an
127     ↪ arc) using set
128     // PID/Slew gains, and executing the AsyncActions at the right times
129
130 /* ----- Path Following Methods ----- */
131 static void simpleFollow(
132     SimplePath ipath, QLength ilookDist = 6_in, std::vector<AsyncAction> iactions =
133     ↪ {},
134     PID imagitudePID = PID(0.4, 0.005, 2.6, 0.5, 0.25, 0.05, 10_ms),
135     PID iturnPID = PID(0.028, 0.0, 0.08, 0.0, 1.5, 0.1, 10_ms),
136     Slew imagitudeSlew = Slew(1000, 1000),
137     Slew iturnSlew = Slew(1000,
138     ↪ 1000)); // follows the path, ipath using set lookahead
139     ↪ distance
140     // (ilookDist) and PID/Slew gains while executing
141     ↪ the
142     // AsyncActions at the right times (only on the
143     ↪ last segment)
144
145 /* ----- Motion Profiling ----- */
146 static std::shared_ptr<AsyncMotionProfileController>
147     mprofiler; // okapi motion profile controller
148 static void generatePathToPoint(

```

```

136     PathfinderPoint ipoint,
137     const std::string & iname); // use Pathfinder through okapi to make a motion
    ↪ profile
138     static void followPathfinder(const std::string & iname, bool ibackwards = false,
139                                 bool imirrored = false); // follow Pathfinder path
    ↪ through okapi
140     static void followTraj(Trajectory & itraj); // follow trajectory loaded from sd card
141 };
142
143 namespace def
144 {
145     extern Drivetrain drivetrain; // declares the drivetrain object as extern, to make sure
    ↪ it only gets
146                                     // constructed once
147 }

```

2.7 include/movement/paths/ProfileStep.hpp

```
1  /**
2   * ProfileStep.hpp
3   *
4   * ProfileStep is used for organizing the information parsed from
5   * motion profiles stored on the sd card, calculated by the
6   * publically available GitHub repository, TrajectoryLib by
7   * Team254 (FRC Team 254, The Cheesy Poofs), found here:
8   * https://github.com/Team254/TrajectoryLib. The trajectories
9   * are calculated on a computer, and stored on the sd card for
10  * the robot to use. Each time step of the profile is read from
11  * the sd card, and stored in an instance of ProfileStep by the
12  * Trajectory class.
13  */
14  #pragma once
15  #include "main.h"
16  struct ProfileStep
17  {
18      float pos{0.000};
19      float vel{0.000};
20      float acc{0.000};
21      float jerk{0.000};
22      float heading{0.000};
23      float dt{0.000};
24      float x{0.000};
25      float y{0.000};
26
27      const std::string getString(); // return the contents of the ProfileStep for
    ↪ testing purposes
28  };
```

2.8 include/movement/paths/SimplePath.hpp

```
1  /**
2   * SimplePath.hpp
3   *
4   * SimplePath is a simple struct that has a list of points
5   * on a path represented by ExtendedPoints in a std::vector.
6   * This is used for path following by the Drivetrain class.
7   */
8  #pragma once // makes sure the file is only included once
9  #include "main.h" // gives access to objects declared elsewhere (std::vector and
   ↳ ExtendedPoint)
10 struct SimplePath
11 {
12     std::vector<ExtendedPoint> mpoints;
13
14     ExtendedPoint at(size_t iindex); // gets the point at iindex
15     ExtendedPoint last(); // gets the last point
16     int size(); // number of points
17 };
```

2.9 include/movement/paths/Trajectory.hpp

```
1  /**
2   * Trajectory.hpp
3   *
4   * This file contains the declaration of the Trajectory class.
5   * The Trajectory class reads and stores motion profile information
6   * from the sd card. Motion profiles stored on the sd card are
7   * calculated by the publically availible GitHub repository,
8   * TrajectoryLib by Team254 (FRC Team 254, The Cheesy Poofs),
9   * found here: https://github.com/Team254/TrajectoryLib. The
10  * trajectories are calculated on a computer, and stored on the
11  * sd card for the robot to use. Each time step of the profile is
12  * read from the sd card, and stored in an instance of ProfileStep
13  * by the Trajectory class.
14  *
15  * The paths are intended to be executed by the Drivetrain class.
16  */
17  #pragma once
18  #include "main.h"
19  class Trajectory
20  {
21      double mkP, mkD, mkV, mkA; // control constants
22      int mstepNumber; // index of current step
23      double mlastErrorL, mlastErrorR; // old error values
24
25      std::string mname; // name of the movement from the top of the file
26      int mlength; // number of steps for each side to execute
27      ProfileStep * mleftSteps; // steps for the left side
28      ProfileStep * mrightSteps; // steps for the right side
29
30  public:
31      Trajectory(const char * ifileName, double ikP = 0.0, double ikD = 0.0, double ikV =
32          ↪ 0.025,
33          double ikA = 0.0); // constructor that specifies control constants
34      ~Trajectory(); // destructor that handles dynamically allocated arrays to prevent
35          ↪ memory issues
36
37      std::string getName();
38      int getLength();
39      void reset(); // sets mstepNumber back to 0
40      std::pair<ProfileStep, ProfileStep>
41      getStep(int istepNumber); // get the left and right values at a certain step
42      void setGains(const double ikP, const double ikD, const double ikV, const double
43          ↪ ikA);
44      bool isDone(); // checks to see if all of the steps have been executed
45
46      std::pair<double, double>
47      iterate(const double ileftDistSoFar,
48          const double irlightDistSoFar); // calculate the motor vales at the next step
49  };
50
51 namespace def
52 {
```



```
50 | extern Trajectory traj_test;  
51 | extern Trajectory TestSpline;  
52 | } // namespace def
```

2.10 include/stateMachines/BallControlStateMachine.hpp

```
1  /**
2   * BallControlStateMachine.hpp
3   *
4   * This file contains the definitions of the BallControlStateMachine class.
5   * BallControlStateMachine inherits from VStateMachine, and
6   * it is responsible for controlling all of the ball manipulators
7   * (intake, indexer, filter, and flywheel).
8   *
9   * The intake, indexer, and flywheel all have their own mini
10  * state machine in structs all contained in
11  * BallControlStateMachine. BallControlStateMachine puts them
12  * all together to make them function cohesively
13  */
14  #pragma once // makes sure the file is only included once
15  #include "main.h" // gives access to dependencies from other files
16  class BallControlStateMachine : public VStateMachine // state machine to represent the
17  ↪ ball
18
19                                     // controllers
20                                     ↪ (intake/indexer/flywheel)
21  {
22  public:
23      BallControlStateMachine(); // constructor to set defaults
24
25      void controlState() override; // sets the state based on inputs from the controller
26      void update() override; // controls the robot based on the state
27
28      void itIn(); // spins the intakes in
29      void itOut(); // spins the intakes out
30      void itOff(); // stops the intakes
31      void ixUp(); // spins the indexer up
32      void ixDown(); // spins the indexer down
33      void ixOff(); // stops the indexer
34      void fwShoot(); // shoots the flywheel
35      void fwFilter(); // spins the flywheel backwards
36      void fwOff(); // stops the flywheel
37
38      void itInFor(double imilliseconds); // spins the intakes for specified number of
39      ↪ milliseconds
40      void ixUpFor(double imilliseconds); // spins the indexer for specified number of
41      ↪ milliseconds
42      void shoot(int ims = 300); // shoots a ball
43
44      bool controlEnabled; // decides if the state machine should pay attention to the
45      ↪ controller
46
47  private:
48      /* ----- Controls ----- */
49      ControllerButton & mbtnIn;
50      ControllerButton & mbtnOut;
51      ControllerButton & mbtnShoot;
52      ControllerButton & mbtnFilter;
```

```

48  /* ----- Nested Classes ----- */
49  struct MIntake // controls the intakes
50  {
51      MIntake(); // constructor to set defaults
52
53      enum MStates // enumeration to organize all possible states
54      {
55          off,
56          in,
57          out
58      };
59
60      void update(); // updates the subsystem based on the state
61
62      /* ----- State ----- */
63      MStates mstate;
64
65      /* ----- Other ----- */
66      MotorGroup mmotors;
67  } mintake;
68
69  struct MIndexer // controls the indexer
70  {
71      MIndexer(); // constructor to set defaults
72
73      enum class MStates // enumeration to organize all possible states
74      {
75          off,
76          in,
77          out
78      };
79
80      void update(); // updates the subsystem based on the state
81
82      /* ----- State ----- */
83      MStates mstate;
84
85      /* ----- Other ----- */
86      Motor mmotor;
87  } mindexer;
88
89  struct MFlywheel // controls the flywheel
90  {
91      MFlywheel(); // constructor to set defaults
92
93      enum class MStates // enumeration to organize all possible states
94      {
95          off,
96          shoot, // forward
97          filter // reverse
98      };
99
100     void update(); // updates the subsystem based on the state
101

```

```

102      /* ----- State ----- */
103      MStates mstate;
104
105      /* ----- Other ----- */
106      MotorGroup mmotors;
107      } mflywheel;
108 };
109
110 namespace def
111 {
112     extern BallControlStateMachine
113         sm_bc; // declares the sm_bc object as extern, to make sure it only gets
114             ↳ constructed once
115 } // namespace def

```

2.11 include/stateMachines/DrivetrainStateMachine.hpp

```

1  /**
2   * DrivetrainStateMachine.hpp
3   *
4   * This file contains the declaration of the DrivetrainStateMachine class.
5   * DrivetrainStateMachine is a state machine that inherits from VStateMachine.
6   * It has an enumeration of different possible states to make it easy for
7   * the user to controll the drivetrain.
8   *
9   * To use the state machine in auton, you use doAutonMotion() to disable
10  * the normal state machine tasks and run the specified action.
11  */
12  #pragma once // makes sure the file is only included once
13  #include "main.h" // gives access to dependancies from other files
14  class DrivetrainStateMachine : public VStateMachine // state machine to represent the
15  ↪ drivetrain
16  {
17  public:
18      DrivetrainStateMachine(); // constructor to set defaults
19      enum class MStates // enumeration to organize possible states
20      {
21          off,
22          busy, // doing an AutonMotion
23          manual, // standard split arcade drive
24          fieldCentric // standard split arcade, but with field centric turns
25      };
26      MStates getState();
27      void setState(MStates istate);
28
29      void
30      doAutonMotion(std::function<void()> iaction); // disable manual control, and
31      ↪ execute the action
32
33      void controlState() override; // update the state based on controller input
34      void update() override; // move the robot based on the state
35
36  private:
37      /* ----- State ----- */
38      MStates mstate, mlastState;
39
40      bool stateChanged(); // returns whether the last state is the same as the current
41      ↪ one
42
43      /* ----- Controls ----- */
44      Controller & mcontroller; // reference to the controller to get joystick values
45      ControllerButton &
46      mtoggleFieldCentric; // reference to the button that toggles field centric
47      ↪ control
48
49      /* ----- Other ----- */
50      Drivetrain & mdrivetrain; // reference to the drivetrain to give control of the
51      ↪ drivetrain to
52
53      // the state machine

```

```
48 |};
49 |
50 | namespace def
51 | {
52 |   extern DrivetrainStateMachine
53 |     sm_dt; // declares the sm_dt object as extern, to make sure it only gets
54 |     ↪ constructed once
    } // namespace def
```

2.12 include/stateMachines/VStateMachine.hpp

```
1  /**
2   * VStateMachine.hpp
3   *
4   * This file contains the parent class, VStateMachine.
5   * VStateMachine is an abstract base class for all state
6   * machines. It specifies that all state machines should
7   * have a method that controls the state based on user
8   * input, and a method that moves the robot based on the state.
9   */
10 #pragma once // makes sure the file is only included once
11 class VStateMachine // abstract state machine base class
12 {
13     public:
14         virtual void controlState() = 0; // changes the state based on user input
15         virtual void update() = 0; // controls the subsystem based on the current state
16 };
```

2.13 include/util/CustomOdometry.hpp

```
1  /**
2   * CustomOdometry.hpp
3   *
4   * This file contains the declaration of the CustomOdometry class.
5   * CustomOdometry is responsible for doing all the math and storing
6   * information about the robot's position and orientation. Everything
7   * is static, because there doesn't need to be more than one position
8   * calculation.
9   */
10 #pragma once           // makes sure the file is only included once
11 #include "main.h"      // give access to dependencies from other files
12 class CustomOdometry // class that organizes the sensors, calculations, and state
13 {
14     ↪ variables for odometry
15
16     /* ----- Constants ----- */
17     static const double & moffFIn; // offset of forward tracking wheel in inches
18     static const double & moffSIn; // offset of side tracking wheel in inches
19     static const double & mcircIn; // tracking wheel circumference in inches
20
21     /* ----- Sensor References ----- */
22     static ADIEncoder & meF; // left tracking wheel encoder
23     static ADIEncoder & meS; // right tracking wheel encoder
24     static pros::Imu & mimu1; // inertial sensors
25     static pros::Imu & mimu2;
26
27     /* ----- Variables ----- */
28     static OdomState mstate; // position of the robot
29     static bool menabled;    // whether or not the loop is allowed to run
30
31     /* ----- Methods ----- */
32     static std::valarray<double> getSensorVals(); // returns new sensor values
33     friend void odomTaskFunc(void *);             // task to be run all the time.
34
35 public:
36     static OdomState getState(); // returns the current state of the
37     ↪ robot
38     static QLength getX();       // returns the x value of the state
39     static QLength getY();       // returns the y value of the state
40     static QAngle getTheta();    // returns the theta value of the
41     ↪ state
42     static void setState(const OdomState & istate); // sets the state of the robot
43
44     static void enable(); // allows the odometry thread to be started (but does not
45     ↪ start it)
46     static void disable(); // stops the odometry thread from running, prevents it from
47     ↪ starting
48
49     static OdomState mathStep(std::valarray<double> ivalDiff); // does one iteration
50     ↪ of odometry math, given sensor changes
51 };
52
```



```
47 namespace def
48 {
49 extern CustomOdometry customOdom; // declares the customOdom object as extern, to make
   ↳ sure it only gets constructed once
50 }
51
52 void odomTaskFunc(void *); // friend function to CustomOdometry to be run as a separate
   ↳ thread
```

2.14 include/util/definitions.hpp

```
1  /**
2   * definitions.hpp
3   *
4   * This file contains various declarations and definitions for
5   * motors, sensors, controls, constants, and settings, so that
6   * things that might need to be changed are all in one place.
7   */
8  #pragma once // makes sure the file is only included once
9  #include "main.h" // gives access to dependancies from other files
10
11  #define DT_STATES DrivetrainStateMachine::MStates
12  #define IT_STATES BallControlStateMachine::MIntake::MStates
13  #define IX_STATES BallControlStateMachine::MIndexer::MStates
14  #define FW_STATES BallControlStateMachine::MFlywheel::MStates
15
16  #define makeFunc(i) [&]() i
17  #define cutDrive(i)
18      ↪      ↪
19      ↪      ↪
20      ↪      ↪ AsyncAction(i, makeFunc({ def::drivetrain.disable(); })))
21      ↪      ↪
22      ↪      ↪
23      ↪      ↪
24      ↪      ↪
25      ↪      ↪
26      ↪      ↪
27      ↪      ↪
28      ↪      ↪
29      ↪      ↪
30      ↪      ↪
31      ↪      ↪
32      ↪      ↪
33      ↪      ↪
34      ↪      ↪
35      ↪      ↪
36      ↪      ↪
37      ↪      ↪
38      ↪      ↪
39      ↪      ↪
40      ↪      ↪
41      ↪      ↪
42      ↪      ↪
43      ↪      ↪
44      ↪      ↪
45      ↪      ↪
46      ↪      ↪
47      ↪      ↪
48      ↪      ↪
49      ↪      ↪
```

```

50 extern Controller controller;
51
52 /* ----- Drivetrain ----- */
53 extern ControllerButton btn_dt_tglFieldCentric;
54
55 /* ----- Ball Control ----- */
56 extern ControllerButton btn_bc_in;
57 extern ControllerButton btn_bc_out;
58 extern ControllerButton btn_bc_shoot;
59 extern ControllerButton btn_bc_down;
60
61 /* ----- Constants ----- */
62 /* ----- Constants ----- */
63 /* ----- Constants ----- */
64
65 /* ----- Tracking Wheels ----- */
66 const QLength TRACK_WHEEL_DIAMETER = 2.847_in;
67 const QLength TRACK_WHEEL_CIRCUMFERENCE = TRACK_WHEEL_DIAMETER * M_PI;
68 const QLength TRACK_FORWARD_OFFSET = 2.3_in;
69 const QLength TRACK_SIDE_OFFSET = 7_in;
70
71 /* ----- Drivetrain ----- */
72 const QLength DRIVE_WHEEL_DIAMETER = 4.041_in;
73 const double DRIVE_WHEEL_DIAMETER_IN = DRIVE_WHEEL_DIAMETER.convert(inch);
74 const QLength DRIVE_WHEEL_CIRCUMFERENCE = DRIVE_WHEEL_DIAMETER * M_PI;
75 const double DRIVE_WHEEL_CIRCUMFERENCE_IN = DRIVE_WHEEL_CIRCUMFERENCE.convert(inch);
76 const QLength DRIVE_OFFSET = 42_in;
77
78 const QAcceleration DRIVE_MAX_ACCEL = 1_G; // approximate measured linear acceleration
79 const QSpeed DRIVE_MAX_SPEED = 2.7_mps; // a measured linear velocity
80
81 /* ----- Settings ----- */
82 const double SET_DT_MAX = 1; // default drivetrain max speed (1 is 100%)
83 const OdomState SET_ODOM_START = {0_ft, 0_ft, 0_deg}; // starting position of the robot
84 ↳ on the field
85 } // namespace def

```

2.15 include/util/util.hpp

```

1  /**
2   * util.hpp
3   *
4   * This file contains miscellaneous utility functions and classes
5   * to help with the general organization of the rest of the code.
6   */
7  #pragma once          // makes sure the file is only included once
8  #include "main.h"    // gives access to dependancies from other files
9
10 /* ----- */
11 /*                               ExtendedPoint Struct                               */
12 /* ----- */
13 * ExtendedPoint struct inherits from the built in okapi Point struct,
14 * but provides additional functionality, like an orientation value
15 * (theta) as well as x and y values. It also adds some vector operations.
16 */
17 struct ExtendedPoint : Point
18 {
19     ExtendedPoint(QLength ix, QLength iy, QAngle itheta);
20
21     QAngle theta{0_deg}; // stores the orientation at the point, with a default of 0
22     ↪ degrees
23
24     /* ----- Subtraction ----- */
25     ExtendedPoint operator-(const ExtendedPoint & ivec);
26     ExtendedPoint sub(const ExtendedPoint & ivec);
27
28     /* ----- Addition ----- */
29     ExtendedPoint operator+(const ExtendedPoint & ivec);
30     ExtendedPoint add(const ExtendedPoint & ivec);
31
32     /* ----- Multiplication ----- */
33     QLength dot(const ExtendedPoint & ivec); // dot multiply the vectors
34     ExtendedPoint operator*(const double iscalar);
35     ExtendedPoint scalarMult(const double iscalar); // multiply the vectors by a
36     ↪ scalar
37     ExtendedPoint operator*(const ExtendedPoint & ivec); // elementwise multiplication
38     ExtendedPoint eachMult(const ExtendedPoint & ivec);
39
40     /* ----- Comparative ----- */
41     bool operator==(const ExtendedPoint & ipoint); // checks to see if the points are
42     ↪ the same
43
44     /* ----- Other ----- */
45     QLength dist(const ExtendedPoint & ivec); // distance between points
46     QLength mag(); // magnitude
47     ExtendedPoint normalize(); // creates a vector with a length of 1
48     std::string string();
49 };
50
51 /* ----- */
52 /*                               Misc Functions                               */
53 /* ----- */

```

```

50  /* ----- */
51  void waitForImu(); // blocks execution of the code until the imu is done calibrating
52
53  /* ----- OdomDebug Helpers ----- */
54  void odomSetState(OdomDebug::state_t istate); // sets the state of odometry based on
55     ↳ display inputs
56  void odomResetAll(); // resets everything having to do with
57     ↳ odometry (for "Reset" button)
58
59  /* ----- Task Functions ----- */
60  void sm_dt_task_func(void *); // state machine drivetrain task to be run independently
61  void sm_bc_task_func(void *); // state machine ball control task to be run independently
62
63  void display_task_func(void *); // display task to be run independently
64
65  /* ----- Macros ----- */
66  void deploy(); // deploys the robot
67
68  /* ----- Control ----- */
69
70  /* ----- PID Class ----- /
71  * PID is a feedback loop that uses the difference between the goal
72  * and the current position (error) of the robot to decide how much
73  * power to give the motors. The "P" stands for "proportional", and
74  * it adds power proportional to the error, so it gets slower and
75  * slower as it gets closer to the goal to prevent it from driving
76  * too fast passed it. The "D" stands for "derivative", because it
77  * uses the derivative of the error (the speed of the robot) to apply
78  * power. If the robot is moving too fast, the "D" term will slow
79  * down, and if it is moving too slow, the "D" term will speed up.
80  * The "I" stands for "integral", because it uses the integral of the
81  * error (the absement of the robot) to apply power. When the robot
82  * is close to the goal, sometimes the "P" and "D" terms do not
83  * apply enough power to move the robot, but when the robot isn't
84  * moving, the "I" term is accumulating, so it eventually builds up
85  * enough to get the robot even closer to the goal. This implementation
86  * of PID only enables the "I" term when the robot is close enough
87  * to the goal, to prevent "integral windup", which is when the
88  * integral gets too big when it's too far away from the goal.
89  *
90  * We have a PID controller class, because we use different PID loops
91  * in many different places in the code, so we wanted to be able to
92  * be able to quickly make one with constants specific to the application.
93  */
94  class PID
95  {
96      double msettlerError, msettlerDerivative; // target error and derivative for the
97         ↳ settler
98      QTime msettlerTime; // target time for the settler
99      std::unique_ptr<SettledUtil> msettler; // okapi settler that is used to
100         ↳ determine if the PID should stop, based on error, derivative, and time

```

```

100     double mkP, mkI, mkD, mkIRange; // constants
101
102     double merror, mlastError, mtotalError;
103     double mderivative; // used for storing derivative before lastError is overwritten
104
105 public:
106     PID(double ikP, double ikI, double ikD, double ikIRange, double isettlerError,
107         ↪ double isettlerDerivative, QTime isettlerTime); // constructor
108
109     PID(const PID & iothor); // copy constructor
110
111     double getLastError();
112     double getTotalError();
113
114     void setGains(double ikP, double ikI, double ikD);
115
116     double getP();
117     double getI();
118     double getD();
119
120     double iterate(double ierror); // goes through one iteration of the PID loop
121
122     bool isSettled(); // returns whether or not the controller is settled at the target
123 };
124
125 /* ----- Slew Class ----- */
126 * Slew rate control is a system that limits the change in speed to
127 * prevent wheel slip. If the robot changes speed to fast, the wheels
128 * can slip, and make the robot's motion less fluid. When the target
129 * speed changes by a lot, the slew rate controller slowly increases
130 * it's output to eventually get to the target speed.
131 * This Slew rate controller is also intended to be used with PID, but
132 * sometimes when slew is used with PID, it interferes with the settling
133 * of the PID. To prevent this, the slew rate controller is only active
134 * when there are large changes in the target input value, making it only
135 * really affect the beginning of the motion. For example, if the motors
136 * aren't moving, and the target value suddenly jumps to 100%, the slew
137 * controller might gradually increase by increments of 5% until it
138 * reaches 100%, but if the target value jumps to from 0% to 20%, the
139 * slew controller might not engage (actual values depend on constants
140 * "mincrement" and "mactiveDifference").
141 */
142 class Slew
143 {
144     double mincrement; // amount to change between each iteration
145     double mactiveDifference; // threshold to activate slew
146     double mlastValue; // previous value
147
148 public:
149     Slew(double iincrement, double iactiveDifference); // constructor
150
151     double getIncrement();
152     double getActiveDifference();

```

```

153     double getLastValue();
154
155     double iterate(double ivalue); // limits the input value to maximum changes
    ↪ described by constants when run in a loop
156 };
157
158 /* ----- */
159 /*                               Util                               */
160 /* ----- */
161 * The util namespace is used to organaize basic functions that don't
162 * necessarily need to be used for robotics.
163 */
164 namespace util
165 {
166
167 /* ----- DEMA Filter ----- */
168 * DEMA is short for Double Exponential Moving Average. It is a method
169 * is a type of filter that smooths data and gives more weight to
170 * more recent values.
171 *
172 * A Simple Moving Average (SMA) takes the mean of a certain number
173 * of values over a specified period of time. An Exponential Moving
174 * Average (EMA) is similar, but it gives more weight to newer values,
175 * so it more closly tracks the actual value. A DEMA is the EMA of
176 * an EMA. More specifically, it is calculated by 2EMA - EMA(EMA),
177 * and it gives even more weight to newer values.
178 *
179 * The DEMAFilter class was originally added as an easy way to improve
180 * the quality of angle measurements from the inertial sensor. It was
181 * needed because the odometry loop updates at 100hz, and the inertial
182 * sensors used to only update at 50hz. The DEMA filter did improve
183 * the position calculation a small amount, but now the inertial
184 * sensor can update at 100hz (maybe more; it's unclear), and the
185 * filter is no longer useful.
186 */
187 template <int N> // the DEMA filter can be set to use the previous N values, changing
    ↪ how significant newer values are
188 class DEMAFilter
189 {
190     const double mk; // weighting constant
191     double mlastEMA, mlastEMAEMA; // previous EMA values
192
193     double EMACalc(double & inext, double & iold) { return (inext - iold) * mk + iold;
    ↪ } // EMA calculation
194
195 public:
196     DEMAFilter(std::array<double, 2 * N - 1> ifirstVals) : mk(2.0 / (N + 1)) // to
    ↪ start filtering values, the DEMA filter needs to have pre-filtered values. The
    ↪ constructor calculates the "last" values of the EMA and EMA(EMA)
197     {
198         for (int i = 0; i < N; i++) // calc sum of the first N numbers
199         {
200             mlastEMA += ifirstVals[i];
201         }

```

```

202     mlastEMA /= N; // store the SMA (mean) of the first N numbers
203
204     mlastEMAEMA = mlastEMA;
205     for (int i = 0; i < N - 1; i++)
206     {
207         mlastEMA = EMACalc(iffirstVals[i + N], mlastEMA); // put the next values
208         ↳ from the input through EMA filter
209         mlastEMAEMA += mlastEMA; // store sum of these
210         ↳ values
211     }
212     mlastEMAEMA /= N; // store the SMA (mean) of the first N values from the EMA
213 }
214
215 double filter(double iinput) // filters the input value by doing DEMA calculation
216 {
217     double EMA = EMACalc(iinput, mlastEMA); // first EMA
218     double EMAEMA = EMACalc(EMA, mlastEMAEMA); // EMA of first EMA (EMA(EMA))
219     mlastEMA = EMA; // store previous EMA
220     mlastEMAEMA = EMAEMA; // store previous EMA(EMA)
221
222     return 2 * EMA - EMAEMA; // 2EMA - EMA(EMA)
223 }
224 };
225
226 /* ----- Angle Wrappers ----- /
227 * All of these functions take an angle as an input, and return an
228 * angle fitting into a certain range. For example, wrapDeg(370) would
229 * return 10, and wrapDeg180(200) would return -160
230 */
231 double wrapDeg(double iangle); // [0, 360)
232 double wrapDeg180(double iangle); // [-180, 180)
233 double wrapRad(double iangle); // [0, 2pi)
234 double wrapRadPI(double iangle); // [-pi, pi)
235 QAngle wrapQAngle(QAngle iangle); // [0_deg, 360_deg)
236 QAngle wrapQAngle180(QAngle iangle); // [-180_deg, 180_deg)
237
238 /* ----- Find Max ----- /
239 * these functions all find the maximum value of a few different types
240 * of inputs. They use templates so they can be used on different types,
241 * and on arrays of different lengths.
242 */
243 template <class T, std::size_t N>
244 T findMax(const std::array<T, N> && iarray) // returns the max value in iarray
245 {
246     T largest = iarray.at(0); // gives largest a value to compare with
247     for (const T & val : iarray) // loops through all values
248         if (val > largest)
249             largest = val; // stores the largest value
250     return largest;
251 }
252
253 template <class T, std::size_t N>
254 T findMax(const std::array<T, N> & iarray) // returns the max value in iarray
255 {
256     T largest = iarray.at(0); // gives largest a value to compare with

```



```

254     for (const T & val : iarray) // loops through all values
255         if (val > largest)
256             largest = val; // stores the largest value
257     return largest;
258 }
259 template <class T, std::size_t N>
260 T findAbsMax(const std::array<T, N> && iarray) // returns the max absolute value in
    ↪ iarray
261 {
262     T largest = iarray.at(0); // gives largest a value to compare with
263     for (const T & val : iarray) // loops through all values
264         if (abs(val) > largest)
265             largest = abs(val); // stores the largest value
266     return largest;
267 }
268 template <class T, std::size_t N>
269 T findAbsMax(const std::array<T, N> & iarray) // returns the max absolute value in
    ↪ iarray
270 {
271     T largest = iarray.at(0); // gives largest a value to compare with
272     for (const T & val : iarray) // loops through all values
273         if (abs(val) > largest)
274             largest = abs(val); // stores the largest value
275     return largest;
276 }
277
278 /* ----- Fitters ----- /
279  * These functions modify the input to fit in a specified range
280  */
281 template <std::size_t N>
282 std::array<double, N> scaleToFit(double imagnitude, std::array<double, N> && iarray) //
    ↪ scales all elements in iarray to fit within [-imagnitude, imagnitude]
283 {
284     double largest = findAbsMax<double, N>(iarray);
285     if (largest > imagnitude) // if anything is out of range
286     {
287         largest = std::abs(largest);
288         for (double & val : iarray) // scales everything down to fit in the range
289             val = val / largest * imagnitude;
290     }
291     return iarray;
292 }
293 template <std::size_t N>
294 void scaleToFit(double imagnitude, std::array<double, N> & iarray) // scales all
    ↪ elements in iarray to fit within [-imagnitude, imagnitude]
295 {
296     double largest = findAbsMax<double, N>(iarray);
297     if (largest > imagnitude) // if anything is out of range
298     {
299         largest = std::abs(largest);
300         for (double & val : iarray) // scales everything down to fit in the range
301             val = val / largest * imagnitude;
302     }
303 }

```

```

304
305 template <class T, std::size_t N>
306 void chop(T imin, T imax, std::array<T, N> & iarray) // if any values in iarray are out
↳ of range, they are set to the limit
307 {
308     for (double & val : iarray)
309     {
310         if (val > imax)
311             val = imax;
312         else if (val < imin)
313             val = imin;
314     }
315 }
316 template <class T>
317 void chop(T imin, T imax, T & inum) // if the value is out of range, it is set to the
↳ limit
318 {
319     if (inum > imax)
320         inum = imax;
321     else if (inum < imin)
322         inum = imin;
323 }
324 } // namespace util

```

3 Source Files

3.1 src/Auton.cpp

```
1  /**
2   * Auton.cpp
3   *
4   * This contains the definitions of the Auton struct,
5   * which is responsible for reading the sd card to determine
6   * which auton is selected, and running the correct auton.
7   */
8  #include "main.h" // gives access to Auton and other dependencies
9
10 Auton::Autons Auton::auton =
11     Auton::Autons::none; // default auton is none, if the sd card is not installed
12
13 void Auton::readSettings() // read the sd card to set the settings
14 {
15     FILE * file; // cpp a file object to be used later
16     if (pros::usd::is_installed()) // checks if the sd card is installed before trying
17         ↪ to read it
18     {
19         file = fopen("/usd/auton_settings.txt", "r"); // open the auton settings
20         if (file) // check to see if the file opened correctly
21         {
22             fscanf(file, "%i", &auton);
23         }
24         else
25         {
26             std::cout << "/usd/auton_settings.txt is null."
27                 << std::endl; // if the file didn't open right, tell the terminal
28         }
29         fclose(file); // close the file
30     }
31 }
32
33 void Auton::runAuton() // runs the selected auton
34 {
35     pros::Task auton_task(auton_task_func);
36 }
37
38 /*
39  \---/
40  / 1 \
41  \---/
42
43 */
44
45 /* ----- */
46 /* Private Information */
47 /* ----- */
48 void Auton::auton_task_func(void *) // separate thread for running the auton, in case a
    ↪ particular
    // auton needs control over its thread
```

```

49 {
50 // when making autons, you must add the text to the dropdown in DisplayControl.cpp,
51 ↪ a new enum
52 // value in Auton.hpp, and a new case is this switch
53 switch (auton)
54 {
55     case Autons::none:
56         break;
57     case Autons::test:
58         def::sm_dt.doAutonMotion(makeFunc({
59             def::drivetrain.strafeToPoint({4_ft, 0_ft, 0_deg});
60             def::drivetrain.strafeToPoint({4_ft, -3.8_ft, 180_deg});
61             def::drivetrain.strafeToPoint({4_in, -3.8_ft, 0_deg});
62             def::drivetrain.strafeToPoint({4_in, -4_in, 0_deg});
63         }));
64         break;
65     case Autons::prog:
66         /**
67         ↪ * `makeFunc()` is a preprocessor macro that takes the contents and turns
68         ↪ them into a
69         ↪ * lambda function, [](){}. It is frequently used to specify the actions in
70         ↪ * AsyncActions.
71         ↪ *
72         ↪ * `cutDrive()` is a preprocessor macro that adds an AsyncAction to the
73         ↪ motion that
74         ↪ * disables the motion at a certain error (in inches) from the target. This
75         ↪ is used
76         ↪ * frequently because many motions do not need to go exactly to the target.
77         ↪ When
78         ↪ * possible, stopping the motion before it reaches the target is faster,
79         ↪ because it
80         ↪ * doesn't need to use PID to settle.
81         ↪ */
82         CustomOdometry::setState({8_in, 37_in, 0_deg}); // set the starting position
83         def::sm_dt.doAutonMotion(makeFunc({
84             deploy(); // deploys the hood
85             /* ----- Goal 1 ----- */
86             def::sm_bc.ixUp(); // start the indexer to get the first ball
87             def::drivetrain.strafeToPoint(
88                 {17_in, 36_in, 0_deg}, // get the first ball (now 2 balls)
89                 cutDrive(2));
90             def::drivetrain.strafeToPoint({15_in, 16_in, 225_deg}, // line up with
91                 ↪ goal #1
92                 cutDrive(1.5));
93             def::sm_bc.ixOff(); // turns off the indexer
94             pros::delay(300);
95             def::sm_bc.shoot(); // now 1 ball
96
97             /* ----- Goal 2 ----- */
98             def::drivetrain.strafeToPoint(
99                 {35_in, 24_in, -90_deg}, // line up with the second ball
100                 cutDrive(2));
101             def::sm_bc.ixUp(); // get ready for the next ball by starting the
102                 ↪ indexer

```

```

95     def::drivetrain.strafeToPoint({35_in, 10_in, -90_deg}); // get the next
96     ↪ ball (now 2)
97     def::drivetrain.strafeToPoint({73_in, 26_in, 0_deg}, // get the next
98     ↪ ball (now 3)
99
100         cutDrive(2));
101     def::drivetrain.strafeToPoint(
102         {73_in, 29_in, -90_deg}, {}, PID(0.4, 0.005, 2.6, 0.5, 0.5, 0.5,
103         ↪ 1_ms),
104         PID(0.028, 0.0, 0.08, 0.0, 5, 2,
105         ↪ 1_ms)); // turn to face the goal with custom PID gains, because
106         ↪ for some
107         ↪ // reason, this specific motion frequently settled
108         ↪ inconsistantly
109     def::drivetrain.strafeToPoint({71_in, 20_in, -90_deg}, // drive to the
110     ↪ next goal
111
112         cutDrive(1));
113     def::sm_bc.ixOff(); // turns off the indexer
114     def::sm_bc.shoot(); // now 2 balls
115
116     /* ----- Goal 3 ----- */
117     def::drivetrain.strafeToPoint(
118         {108_in, 34_in, 0_deg},
119         {AsyncAction(10,
120         ↪ makeFunc({ def::sm_bc.ixUp(); })), // start the
121         ↪ indexer mid-motion
122         ↪ AsyncAction(2, makeFunc({ def::drivetrain.disable(); })))}; //
123         ↪ same cutDrive(2)
124     def::drivetrain.strafeToPoint({118_in, 34_in, 0_deg}, cutDrive(1));
125     def::drivetrain.strafeToPoint({126_in, 14_in, -36_deg}, // line up with
126     ↪ goal #3
127
128         cutDrive(1));
129     def::sm_bc.ixOff(); // turn off the indexer
130     def::sm_bc.shoot(); // now 2 balls
131
132     /* ----- Goal 4 ----- */
133     def::drivetrain.strafeToPoint({114_in, 72_in, 0_deg}, // move towards
134     ↪ goal #4
135
136         cutDrive(3));
137     def::sm_bc.ixUp();
138     def::drivetrain.strafeToPoint({125_in, 72_in, -2_deg}, // line up with
139     ↪ goal #4
140
141         cutDrive(1));
142     def::sm_bc.ixOff(); // turs the indexer off
143     def::sm_bc.shoot(); // now 1 ball
144     def::sm_bc.ixUp(); // get ready to shoot the next ball
145     pros::delay(700);
146     def::sm_bc.shoot(); // shoot now 0 balls
147     def::sm_bc.ixOff(); // turns the indexer off
148
149     /* ----- Goal 5 ----- */
150     def::drivetrain.strafeToPoint({122_in, 90_in, 90_deg}, // move towards
151     ↪ the next ball
152
153         cutDrive(2));
154     def::sm_bc.ixUp(); // turns the indexer on

```

```

137 def::drivetrain.strafeToPoint(
138     {123_in, 108_in, 90_deg}, // get the next ball (now 1 ball)
139     cutDrive(1));
140 def::drivetrain.strafeToPoint({120_in, 124_in, 45_deg}, // move towards
    ↪ goal #5
141     cutDrive(3));
142 def::drivetrain.strafeToPoint({130_in, 126_in, 43_deg}, // line up with
    ↪ goal #5
143     cutDrive(1));
144 def::sm_bc.shoot(500); // now 0 balls
145 def::sm_bc.ixOff(); // turns the indexer off
146
147 /* ----- Goal 6 ----- */
148 def::sm_bc.ixUp(); // turns the indexer on
149 def::drivetrain.strafeToPoint(
150     {85_in, 120_in, 180_deg}, // move towards the next ball
151     cutDrive(2));
152 def::drivetrain.strafeToPoint({70_in, 118_in, 180_deg}, // get the next
    ↪ ball (now 1)
153     cutDrive(1));
154 def::drivetrain.strafeToPoint({77_in, 123_in, 90_deg},
155     cutDrive(0.25)); // line up with the goal
156 def::sm_bc.shoot(600); // now 0
157 def::sm_bc.ixOff(); // turns off the indexer
158 pros::delay(100); // pause to make sure the shot works
159
160 /* ----- Goal 7 ----- */
161 def::sm_bc.ixUp(); // get ready for the next ball
162 def::drivetrain.strafeToPoint({37_in, 124_in, 90_deg}, // line up with
    ↪ the next ball
163     cutDrive(3));
164 def::drivetrain.strafeToPoint({38_in, 136_in, 90_deg}); // get the next
    ↪ ball (now 1)
165 def::drivetrain.strafeToPoint({17.5_in, 125.5_in, 119_deg}, // line up
    ↪ with goal #7
166     cutDrive(1));
167 def::sm_bc.shoot(600); // now 0 balls
168 def::sm_bc.ixOff(); // turns off the indexer
169
170 /* ----- Goal 8 ----- */
171 def::drivetrain.strafeToPoint({28_in, 129_in, -90_deg}); // turn around
172 def::sm_bc.ixUp(); // turns the indexer on
173 def::drivetrain.strafeToPoint(
174     {28_in, 119_in, -90_deg}, // gets the next ball (now 1)
175     cutDrive(2));
176 def::drivetrain.strafeToPoint({28_in, 69_in, 180_deg}, // move towards
    ↪ goal #8
177     cutDrive(3));
178 def::drivetrain.strafeToPoint({21_in, 72_in, 180_deg}, // line up with
    ↪ goal #8
179     cutDrive(1));
180 def::sm_bc.shoot(600); // now 0
181 def::sm_bc.ixOff();
182

```

```

183      /* ----- Goal 9 ----- */
184      def::drivetrain.strafeToPoint({23_in, 72_in, 0_deg}, cutDrive(0.5)); //
185          ↪ turn around
186      def::sm_bc.ixUp(); // get ready to get the ball by turning the indexer
187          ↪ on
188      def::drivetrain.strafeToPoint({48_in, 72_in, 0_deg}, // get the next
189          ↪ ball (now 1)
190          cutDrive(1));
191      def::drivetrain.strafeToPoint(
192          {36_in, 72_in, 0_deg}, // back up to make sure the descorer doesn't
193          ↪ hit the goal
194          cutDrive(2));
195      def::sm_bc.itOut(); // deploy
196      pros::delay(600); //
197      def::sm_bc.itOff(); // stop the descorer
198      def::drivetrain.strafeToPoint({56_in, 69_in, 0_deg}); // descore
199      pros::delay(500);
200      def::sm_bc.ixOff(); // stop the indexer
201      def::drivetrain.strafeToPoint({30_in, 76_in, 0_deg}, cutDrive(2)); //
202          ↪ back up
203      def::drivetrain.strafeToPoint({57_in, 79_in, -10_deg}, // go to the goal
204          cutDrive(1));
205      def::sm_bc.shoot(); // now 0
206  }));
207  break;
208  }
209  }

```

3.2 src/definitions.cpp

```
1  /**
2   * definitions.cpp
3   *
4   * This file contains various declarations and definitions for
5   * motors, sensors, controls, constants, and settings, so that
6   * things that might need to be changed are all in one place.
7   */
8  #include "main.h" // gives access to definition.hpp and other dependencies
9
10 namespace def
11 {
12     /* ----- */
13     /*                      Devices                      */
14     /* ----- */
15
16     /* ----- Motors ----- */
17     Motor mtr_dt_left_front(16);
18     Motor mtr_dt_right_front(-1);
19     Motor mtr_dt_right_back(-19);
20     Motor mtr_dt_left_back(20);
21     /* ----- */
22     Motor mtr_it_left(18);
23     Motor mtr_it_right(-3);
24     /* ----- */
25     Motor mtr_ix(2);
26     /* ----- */
27     Motor mtr_fw1(-17);
28     Motor mtr_fw2(7);
29
30     /* ----- Sensors ----- */
31     ADIEncoder track_encoder_forward('G', 'H', true);
32     ADIEncoder track_encoder_side('E', 'F', true);
33     pros::Imu imu_top(4);
34     pros::Imu imu_bottom(5);
35
36     /* ----- */
37     /*                      Controls                      */
38     /* ----- */
39     Controller controller = Controller();
40
41     /* ----- Drivetrain ----- */
42     ControllerButton btn_dt_tglFieldCentric = ControllerDigital::A;
43
44     /* ----- Ball Control ----- */
45     ControllerButton btn_bc_in = ControllerDigital::R1;
46     ControllerButton btn_bc_out = ControllerDigital::R2;
47     ControllerButton btn_bc_shoot = ControllerDigital::L1;
48     ControllerButton btn_bc_down = ControllerDigital::L2;
49
50     /* ----- */
51     /*                      Constructs                      */
52     /* ----- */
```



```

53 CustomOdometry customOdom = CustomOdometry(); // object that calculates position
54
55 Drivetrain drivetrain = Drivetrain(); // used by DrivetrainStateMachine for drivetrain
   ↳ control
56
57 DrivetrainStateMachine sm_dt = DrivetrainStateMachine(); // state machine to control
   ↳ the drivetrain
58 BallControlStateMachine sm_bc = BallControlStateMachine(); // state machine for ball
   ↳ manipulators
59 } // namespace def

```

3.3 src/main.cpp

```
1  /**
2   * main.cpp
3   *
4   * This file contains the orchestration of all the compenents. It
5   * starts all of the separate tasks that are needed for controlling
6   * the robot, and has all the functions called by the competition
7   * switch.
8   */
9  #include "main.h" // gives access to dependencies from other files
10
11 DisplayControl def::display = DisplayControl();
12 pros::Task sm_dt_task(sm_dt_task_func);
13 pros::Task sm_bc_task(sm_bc_task_func);
14 pros::Task odomTask(odomTaskFunc);
15 pros::Task display_task(display_task_func);
16
17 /**
18  * Runs initialization code. This occurs as soon as the program is started.
19  *
20  * All other competition modes are blocked by initialize; it is recommended
21  * to keep execution time for this mode under a few seconds.
22  */
23 void initialize()
24 {
25     Auton::readSettings(); // read sd card to remeber the auton selected when the brain
26     ↪ was run last
27     def::display.setAutonDropdown(); // update auton dropdown to match the sd card
28
29     def::mtr_dt_left_front.setEncoderUnits(AbstractMotor::encoderUnits::degrees);
30     def::mtr_dt_right_front.setEncoderUnits(AbstractMotor::encoderUnits::degrees);
31     def::mtr_dt_right_back.setEncoderUnits(AbstractMotor::encoderUnits::degrees);
32     def::mtr_dt_left_back.setEncoderUnits(AbstractMotor::encoderUnits::degrees);
33     def::mtr_it_left.setEncoderUnits(AbstractMotor::encoderUnits::degrees);
34     def::mtr_it_right.setEncoderUnits(AbstractMotor::encoderUnits::degrees);
35     def::mtr_ix.setEncoderUnits(AbstractMotor::encoderUnits::degrees);
36     def::mtr_fw1.setEncoderUnits(AbstractMotor::encoderUnits::degrees);
37     def::mtr_fw2.setEncoderUnits(AbstractMotor::encoderUnits::degrees);
38 }
39
40 /**
41  * Runs while the robot is in the disabled state of Field Management System or
42  * the VEX Competition Switch, following either autonomous or opcontrol. When
43  * the robot is enabled, this task will exit.
44  */
45 void disabled() {}
46
47 /**
48  * Runs after initialize(), and before autonomous when connected to the Field
49  * Management System or the VEX Competition Switch. This is intended for
50  * competition-specific initialization routines, such as an autonomous selector
51  * on the LCD.
52  */
```

```

52  * This task will exit when the robot is enabled and autonomous or opcontrol
53  * starts.
54  */
55  void competition_initialize() {}
56
57  /**
58   * Runs the user autonomous code. This function will be started in its own task
59   * with the default priority and stack size whenever the robot is enabled via
60   * the Field Management System or the VEX Competition Switch in the autonomous
61   * mode. Alternatively, this function may be called in initialize or opcontrol
62   * for non-competition testing purposes.
63   *
64   * If the robot is disabled or communications is lost, the autonomous task
65   * will be stopped. Re-enabling the robot will restart the task, not re-start it
66   * from where it left off.
67   */
68  void autonomous()
69  {
70      Auton::runAuton(); // uses the auton class to run the slected auton
71  }
72
73  /**
74   * Runs the operator control code. This function will be started in its own task
75   * with the default priority and stack size whenever the robot is enabled via
76   * the Field Management System or the VEX Competition Switch in the operator
77   * control mode.
78   *
79   * If no competition control is connected, this function will run immediately
80   * following initialize().
81   *
82   * If the robot is disabled or communications is lost, the
83   * operator control task will be stopped. Re-enabling the robot will restart the
84   * task, not resume it from where it left off.
85   */
86  void opcontrol()
87  {
88
89      def::sm_dt.setState(
90          DT_STATES::manual); // set the drivetrain to basic controls during drivercontrol
91
92      // there is no need for a loop in opcontrol(), because there are already other
93      ↪ tasks running
94      // that control all of the movement
95      // while (true)
96      // {
97      //     pros::delay(20);
98      // }
99  }

```

3.4 src/gui/DisplayControl.cpp

```
1  /**
2   * DisplayControl.cpp
3   *
4   * This file contains the definitions for the DisplayControl class.
5   * DisplayControl is the class that handles the organization of the
6   * LittleV Graphics Library (LVGL) objects on the screen of the brain.
7   */
8  #include "main.h" // gives access to the DisplayControl declaration and other
   ↳ dependencies
9
10 /* ----- Tabview Elements ----- */
11 lv_obj_t * DisplayControl::mtabview = lv_tabview_create(lv_scr_act(), NULL); // creates
   ↳ the tabview
12
13 lv_obj_t * DisplayControl::mtabview_odom = lv_tabview_add_tab(
14     DisplayControl::mtabview,
15     "Odom"); // creates the tab on the screen that shows the calculated robot position
16
17 lv_obj_t * DisplayControl::mtabview_auton = lv_tabview_add_tab(
18     DisplayControl::mtabview, "Auton"); // creates the tab with the auton selection
   ↳ dropdown
19 lv_obj_t * DisplayControl::mtabview_auton_dropdown =
20     lv_ddlist_create(mtabview_auton, NULL); // creates the auton selection dropdown
21 lv_res_t DisplayControl::tabview_auton_dropdown_action(
22     lv_obj_t * idropdown) // specifies the code to be executed when the auton dropdown
   ↳ is changed
23 {
24     FILE * file; // creates an object that will be used to reference the file
   ↳ containing the
25         // selected auton
26     if (pros::usd::is_installed()) // makes sure the sd card is installed before trying
   ↳ to access
27         // its contents
28     {
29         file = fopen("/usd/auton_settings.txt", "w"); // opens the auton settings file
30         if (file) // makes sure the file was accessed correctly
31         {
32             fprintf(file, "%i",
33                 lv_ddlist_get_selected(idropdown)); // update sd card based on new
   ↳ value
34         }
35         else
36         {
37             std::cout
38                 << "/usd/auton_settings.txt is null"
39                 << std::endl; // output to the terminal if the sd card was not accessed
   ↳ correctly
40         }
41         fclose(file);
42         Auton::readSettings(); // update auton based on new sd card values
43     }
44 }
```

```

45     return LV_RES_OK; // required for dropdown callback
46 }
47
48 lv_obj_t * DisplayControl::mtabview_graph = lv_tabview_add_tab(
49     DisplayControl::mtabview, "Graph"); // creates the tab with the graph for debugging
50 lv_obj_t * DisplayControl::mtabview_graph_chart =
51     lv_chart_create(DisplayControl::mtabview_graph, NULL); // create the graph
52
53 // create 7 series of different color, so it is easy to make a graph with any color
54 lv_chart_series_t * DisplayControl::mtabview_graph_chart_series_0 =
55     lv_chart_add_series(DisplayControl::mtabview_graph_chart, LV_COLOR_RED);
56 lv_chart_series_t * DisplayControl::mtabview_graph_chart_series_1 =
57     lv_chart_add_series(DisplayControl::mtabview_graph_chart, LV_COLOR_ORANGE);
58 lv_chart_series_t * DisplayControl::mtabview_graph_chart_series_2 =
59     lv_chart_add_series(DisplayControl::mtabview_graph_chart, LV_COLOR_YELLOW);
60 lv_chart_series_t * DisplayControl::mtabview_graph_chart_series_3 =
61     lv_chart_add_series(DisplayControl::mtabview_graph_chart, LV_COLOR_GREEN);
62 lv_chart_series_t * DisplayControl::mtabview_graph_chart_series_4 =
63     lv_chart_add_series(DisplayControl::mtabview_graph_chart, LV_COLOR_BLUE);
64 lv_chart_series_t * DisplayControl::mtabview_graph_chart_series_5 =
65     lv_chart_add_series(DisplayControl::mtabview_graph_chart, LV_COLOR_PURPLE);
66 lv_chart_series_t * DisplayControl::mtabview_graph_chart_series_6 =
67     lv_chart_add_series(DisplayControl::mtabview_graph_chart, LV_COLOR_MAGENTA);
68
69 lv_obj_t * DisplayControl::mtabview_misc =
70     lv_tabview_add_tab(DisplayControl::mtabview, "Misc"); // creates the miscellaneous
71     ↪ debugging tab
72 lv_obj_t * DisplayControl::mtabview_misc_container =
73     lv_obj_create(DisplayControl::mtabview_misc, NULL);
74 lv_obj_t * DisplayControl::mtabview_misc_label =
75     lv_label_create(mtabview_misc_container, NULL); // creates the left text box
76 lv_obj_t * DisplayControl::mtabview_misc_label_2 =
77     lv_label_create(mtabview_misc_container, NULL); // creates the right text box
78
79 /* ----- Styles ----- */
80 lv_style_t DisplayControl::mstyle_tabview_indic;
81 lv_style_t DisplayControl::mstyle_tabview_btn;
82 lv_style_t DisplayControl::mstyle_tabview_btn_tgl;
83 lv_style_t DisplayControl::mstyle_tabview_btn_pr;
84 lv_style_t DisplayControl::mstyle_tabview_container;
85 lv_style_t DisplayControl::mstyle_text;
86
87 /* ----- Public Information ----- */
88
89 DisplayControl::DisplayControl() : modom(mtabview_odom, LV_COLOR_PURPLE)
90 {
91     /* ----- Style Setup ----- /
92     * Specifies what each style should look like when they are used.
93     */
94     lv_style_copy(&mstyle_tabview_indic, &lv_style_plain);
95     mstyle_tabview_indic.body.padding.inner = 5;
96
97     lv_style_copy(&mstyle_tabview_btn, &lv_style_plain);

```

```

98 mstyle_tabview_btn.body.main_color = LV_COLOR_PURPLE;
99 mstyle_tabview_btn.body.grad_color = LV_COLOR_PURPLE;
100 mstyle_tabview_btn.text.color = LV_COLOR_WHITE;
101 mstyle_tabview_btn.body.border.part = LV_BORDER_BOTTOM;
102 mstyle_tabview_btn.body.border.color = LV_COLOR_WHITE;
103 mstyle_tabview_btn.body.border.width = 1;
104 mstyle_tabview_btn.body.padding.ver = 4;
105
106 lv_style_copy(&mstyle_tabview_btn_tgl, &mstyle_tabview_btn);
107 mstyle_tabview_btn_tgl.body.border.part = LV_BORDER_FULL;
108 mstyle_tabview_btn_tgl.body.border.width = 2;
109
110 lv_style_copy(&mstyle_tabview_btn_pr, &lv_style_plain);
111 mstyle_tabview_btn_pr.body.main_color = LV_COLOR_WHITE;
112 mstyle_tabview_btn_pr.body.grad_color = LV_COLOR_WHITE;
113 mstyle_tabview_btn_pr.text.color = LV_COLOR_WHITE;
114
115 lv_style_copy(&mstyle_tabview_container, &lv_style_plain_color);
116 mstyle_tabview_container.body.main_color = LV_COLOR_PURPLE;
117 mstyle_tabview_container.body.grad_color = LV_COLOR_PURPLE;
118 mstyle_tabview_container.body.border.width = 0;
119 mstyle_tabview_container.body.radius = 0;
120 mstyle_tabview_container.body.padding.inner = 0;
121 mstyle_tabview_container.body.padding.hor = 0;
122 mstyle_tabview_container.body.padding.ver = 0;
123
124 lv_style_copy(&mstyle_text, &lv_style_plain);
125 mstyle_text.text.color = LV_COLOR_WHITE;
126 mstyle_text.text.opa = LV_OPA_100;
127
128 lv_tabview_set_style(mtabview, LV_TABVIEW_STYLE_INDIC,
129                     &mstyle_tabview_indic); // set tabview styles
130 lv_tabview_set_style(mtabview, LV_TABVIEW_STYLE_BTN_REL, &mstyle_tabview_btn);
131 lv_tabview_set_style(mtabview, LV_TABVIEW_STYLE_BTN_PR, &mstyle_tabview_btn_pr);
132 lv_tabview_set_style(mtabview, LV_TABVIEW_STYLE_BTN_TGL_REL,
133                     ↪ &mstyle_tabview_btn_tgl);
134 lv_tabview_set_style(mtabview, LV_TABVIEW_STYLE_BTN_TGL_PR, &mstyle_tabview_btn_pr);
135
136 /* ----- Auton Tab ----- */
137 * When making autons, you must add the text this dropdown, a new
138 * enum value in Auton.hpp, and a new case in the switch in Auton.cpp.
139 */
140 lv_ddlist_set_options(mtabview_auton_dropdown, "none\n"
141                                     "test\n"
142                                     "prog\n"); // auton types in
143                                     ↪ selection dropdown
144
145 lv_ddlist_set_action(mtabview_auton_dropdown,
146                     tabview_auton_dropdown_action); // set the dropdown callback to
147                                     // tabview_auton_dropdown_acti_
148                                     ↪ on()
149
150 lv_obj_align(mtabview_auton_dropdown, NULL, LV_ALIGN_IN_TOP_LEFT, 0,
151             0); // align the dropdown in the top left
152
153 lv_obj_set_style(mtabview_auton, &mstyle_tabview_container); // set styles

```

```

149
150  /* ----- Graph Tab ----- */
151  lv_obj_set_style(mtabview_graph, &mstyle_tabview_container); // set styles
152  lv_obj_set_style(mtabview_graph_chart, &mstyle_tabview_btn_pr);
153
154  lv_page_set_sb_mode(mtabview_graph, LV_SB_MODE_OFF); // hide scrollbar
155
156  lv_chart_set_type(mtabview_graph_chart, LV_CHART_TYPE_LINE); // make chart graph
157  ↪ lines
158  lv_chart_set_point_count(mtabview_graph_chart,
159                          lv_obj_get_width(mtabview_graph_chart) * 2); // set number
160  ↪ of points
161  lv_chart_set_div_line_count(mtabview_graph_chart, 9, 5); // set the number of chart
162  ↪ lines
163  lv_obj_set_size(mtabview_graph_chart, lv_obj_get_width(mtabview_graph),
164                  lv_obj_get_height(mtabview_graph)); // set the graph to fill the
165  ↪ screen
166  lv_obj_align(mtabview_graph_chart, NULL, LV_ALIGN_CENTER, 0, -10); // center chart
167
168  /* ----- Misc Tab ----- */
169  lv_page_set_sb_mode(mtabview_misc, LV_SB_MODE_OFF); // hide scrollbar
170
171  lv_obj_set_style(mtabview_misc, &mstyle_tabview_container); // set styles
172  lv_obj_set_style(mtabview_misc_container, &mstyle_tabview_container);
173  lv_obj_set_size(mtabview_misc_container, lv_obj_get_width(mtabview_misc),
174                  lv_obj_get_height(mtabview_misc)); // set up the background
175  lv_obj_align(mtabview_misc_container, NULL, LV_ALIGN_CENTER, 0, 0);
176  lv_obj_set_style(mtabview_misc_label, &mstyle_text); // set up text boxes (labels)
177  lv_obj_set_style(mtabview_misc_label_2, &mstyle_text);
178
179  lv_label_set_text(mtabview_misc_label, "No data provided."); // set default text
180  ↪ for labels
181  lv_label_set_text(mtabview_misc_label_2, "No data provided.");
182
183  lv_obj_align(mtabview_misc_label, mtabview_misc_container, LV_ALIGN_IN_TOP_LEFT, 0,
184              0); // align labels
185  lv_obj_align(mtabview_misc_label_2, mtabview_misc_container, LV_ALIGN_IN_TOP_RIGHT,
186              -70, 0);
187
188  modom.setStateCallback(
189      odomSetState); // set callbacks for odomDebug to make it interactive on the
190  ↪ screen
191  modom.setResetCallback(odomResetAll);
192  }
193
194  void DisplayControl::setOdomData() // sets the information on the OdomDebug window to
195  ↪ the new
196
197      // calculated odom data
198  {
199      modom.setData({CustomOdometry::getX(), CustomOdometry::getY(),
200                  ↪ CustomOdometry::getTheta()},
201                  {def::track_encoder_forward.get(), def::track_encoder_side.get(),
202                  ↪ 0.0});
203  }

```

```

193 void DisplayControl::setAutonDropdown() // update the auton dropdown to match the sd
    ↳ card
194 {
195     lv_ddlist_set_selected(mtabview_auton_dropdown, (int)Auton::auton);
196 }
197
198 void DisplayControl::setChartData(int iseries,
199                                 double ivalue) // inputs new values to a specific
    ↳ chart series
200 {
201     switch (iseries) // updates the correct series with the new value
202     {
203         case 0:
204             lv_chart_set_next(mtabview_graph_chart, mtabview_graph_chart_series_0,
    ↳ ivalue);
205             break;
206         case 1:
207             lv_chart_set_next(mtabview_graph_chart, mtabview_graph_chart_series_1,
    ↳ ivalue);
208             break;
209         case 2:
210             lv_chart_set_next(mtabview_graph_chart, mtabview_graph_chart_series_2,
    ↳ ivalue);
211             break;
212         case 3:
213             lv_chart_set_next(mtabview_graph_chart, mtabview_graph_chart_series_3,
    ↳ ivalue);
214             break;
215         case 4:
216             lv_chart_set_next(mtabview_graph_chart, mtabview_graph_chart_series_4,
    ↳ ivalue);
217             break;
218         case 5:
219             lv_chart_set_next(mtabview_graph_chart, mtabview_graph_chart_series_5,
    ↳ ivalue);
220             break;
221         case 6:
222             lv_chart_set_next(mtabview_graph_chart, mtabview_graph_chart_series_6,
    ↳ ivalue);
223             break;
224     }
225 }
226
227 void DisplayControl::setMiscData(int ilabel,
228                                 std::string itext) // set the text on text box (label)
    ↳ 1 or 2
229 {
230     if (ilabel == 1)
231     {
232         lv_label_set_text(mtabview_misc_label, itext.c_str());
233     }
234     else if (ilabel == 2)
235     {
236         lv_label_set_text(mtabview_misc_label_2, itext.c_str());

```


237		}
238	}	

3.5 src/gui/odomDebug.cpp

```
1  /**
2   * odomDebug.cpp
3   *
4   * The contents of this file were not written by any members of 333A*.
5   * This is code from the publicly available GitHub repository, odomDebug
6   * by theol0403, found here: https://github.com/theol0403/odomDebug.
7   *
8   * The OdomDebug class is used for the tab on the screen of the brain
9   * that shows the odometry position of the robot in the form of number
10  * values and a moving circle on a picture of the field representing the
11  * robot.
12  *
13  * *slight modifications were made to make it work with the display
14  */
15  #include "main.h"
16
17  /**
18   * @param ix QLength
19   * @param iy QLength
20   * @param itheta QAngle
21   */
22  OdomDebug::state_t::state_t(QLength ix, QLength iy, QAngle itheta) : x(ix), y(iy),
23  ↪   theta(itheta) {}
24
25  /**
26   * @param ix inches
27   * @param iy inches
28   * @param itheta radians
29   */
30  OdomDebug::state_t::state_t(double ix, double iy, double itheta)
31  : x(ix * inch), y(iy * inch), theta(itheta * radian)
32  {
33  }
34
35  /**
36   * @param ileft the left encoder value
37   * @param iright the right encoder value
38   */
39  OdomDebug::sensors_t::sensors_t(double ileft, double iright)
40  : left(ileft), right(iright), hasMiddle(false)
41  {
42  }
43
44  /**
45   * @param ileft the left encoder value
46   * @param iright the right encoder value
47   * @param imiddle the middle encoder value
48   */
49  OdomDebug::sensors_t::sensors_t(double ileft, double iright, double imiddle)
50  : left(ileft), right(iright), middle(imiddle), hasMiddle(true)
51  {
52  }
```

```

52
53 /**
54  * Okapi units that represent a tile (2ft) and a court(12ft)
55  * Literals are `_tl` and `_crt`, respectively
56  */
57 namespace okapi
58 {
59 constexpr QLength tile = 2 * foot;
60 constexpr QLength court = 12 * foot;
61 inline namespace literals
62 {
63 constexpr QLength operator"" _tl(long double x) { return static_cast<double>(x) * tile;
64 ↪ }
65 constexpr QLength operator"" _crt(long double x) { return static_cast<double>(x) *
66 ↪ court; }
67 constexpr QLength operator"" _tl(unsigned long long int x) { return
68 ↪ static_cast<double>(x) * tile; }
69 constexpr QLength operator"" _crt(unsigned long long int x)
70 {
71     return static_cast<double>(x) * court;
72 }
73 } // namespace literals
74 } // namespace okapi
75
76 /**
77  * Constructs the OdomDebug object.
78  * @param parent the lvgl parent, color is inherited
79  */
80 OdomDebug::OdomDebug(lv_obj_t * parent)
81 : OdomDebug(parent, lv_obj_get_style(parent)->body.main_color)
82 {
83 }
84
85 /**
86  * Constructs the OdomDebug object.
87  * @param parent the lvgl parent
88  * @param mainColor The main color for the display
89  */
90 OdomDebug::OdomDebug(lv_obj_t * parent, lv_color_t mainColor)
91 : container(lv_obj_create(parent, NULL))
92 {
93     /**
94      * Container Style
95      */
96     lv_style_copy(&cStyle, &lv_style_plain_color);
97     cStyle.body.main_color = mainColor;
98     cStyle.body.grad_color = mainColor;
99     cStyle.body.border.width = 0;
100     cStyle.body.radius = 0;
101     cStyle.body.padding.inner = 0;
102     cStyle.body.padding.hor = 0;
103     cStyle.body.padding.ver = 0;
104
105     lv_obj_set_style(parent, &cStyle);

```

```

103
104 lv_obj_set_size(container, lv_obj_get_width(parent), lv_obj_get_height(parent));
105 lv_obj_align(container, NULL, LV_ALIGN_CENTER, 0, 0);
106
107 lv_obj_set_style(container, &cStyle);
108
109 /**
110  * Field
111  */
112 lv_obj_t * field = lv_obj_create(container, NULL);
113 fieldDim = std::min(lv_obj_get_width(container), lv_obj_get_height(container));
114 lv_obj_set_size(field, fieldDim, fieldDim);
115 lv_obj_align(field, NULL, LV_ALIGN_IN_RIGHT_MID, 0, 0);
116
117 /**
118  * Field Style
119  */
120 lv_style_copy(&fStyle, &cStyle);
121 fStyle.body.main_color = LV_COLOR_WHITE;
122 fStyle.body.grad_color = LV_COLOR_WHITE;
123 lv_obj_set_style(field, &fStyle);
124
125 /**
126  * Tile Styles
127  */
128 lv_style_copy(&grey, &lv_style_plain);
129 grey.body.main_color = LV_COLOR_HEX(0x828F8F);
130 grey.body.grad_color = LV_COLOR_HEX(0x828F8F);
131 grey.body.border.width = 1;
132 grey.body.radius = 0;
133 grey.body.border.color = LV_COLOR_WHITE;
134
135 lv_style_copy(&red, &grey);
136 red.body.main_color = LV_COLOR_HEX(0xD42630);
137 red.body.grad_color = LV_COLOR_HEX(0xD42630);
138 lv_style_copy(&blue, &grey);
139 blue.body.main_color = LV_COLOR_HEX(0x0077C9);
140 blue.body.grad_color = LV_COLOR_HEX(0x0077C9);
141
142 /**
143  * Tile Layout
144  */
145 std::vector<std::vector<lv_style_t *>> tileData = {
146     {&grey, &red, &grey, &grey, &blue, &grey}, {&red, &grey, &grey, &grey, &grey,
147         ↪ &blue},
148     {&grey, &grey, &grey, &grey, &grey, &grey}, {&grey, &grey, &grey, &grey, &grey,
149         ↪ &grey},
150     {&grey, &grey, &grey, &grey, &grey, &grey}, {&grey, &grey, &grey, &grey, &grey,
151         ↪ &grey}};
152
153 double tileDim = fieldDim / tileData.size(); // tile dimention
154
155 /**
156  * Create tile matrix, register callbacks, assign each tile an ID

```

```

154     */
155     for (size_t y = 0; y < 6; y++)
156     {
157         for (size_t x = 0; x < 6; x++)
158         {
159             lv_obj_t * tileObj = lv_btn_create(field, NULL);
160             lv_obj_set_pos(tileObj, x * tileDim, y * tileDim);
161             lv_obj_set_size(tileObj, tileDim, tileDim);
162             lv_btn_set_action(tileObj, LV_BTN_ACTION_CLICK, tileAction);
163             lv_obj_set_free_num(tileObj, y * 6 + x);
164             lv_obj_set_free_ptr(tileObj, this);
165             lv_btn_set_toggle(tileObj, false);
166             lv_btn_set_style(tileObj, LV_BTN_STYLE_PR, tileData[y][x]);
167             lv_btn_set_style(tileObj, LV_BTN_STYLE_REL, tileData[y][x]);
168         }
169     }
170
171     /**
172      * Robot point using lvgl led
173      */
174     led = lv_led_create(field, NULL);
175     lv_led_on(led);
176     lv_obj_set_size(led, fieldDim / 15 * 2.5, fieldDim / 15 * 2.5);
177
178     lv_style_copy(&ledStyle, &lv_style_plain);
179     ledStyle.body.radius = LV_RADIUS_CIRCLE;
180     ledStyle.body.main_color = LV_COLOR_MAKE(0, 255, 0);
181     ledStyle.body.grad_color = LV_COLOR_MAKE(0, 255, 0);
182     ledStyle.body.border.color = LV_COLOR_WHITE;
183     ledStyle.body.border.width = 2;
184     ledStyle.body.border.opa = LV_OPA_100;
185     lv_obj_set_style(led, &ledStyle);
186
187     /**
188      * Robot line
189      */
190     line = lv_line_create(field, NULL);
191     lv_line_set_points(line, linePoints.data(), linePoints.size());
192     lv_obj_set_pos(line, 0, 0);
193
194     lineWidth = 3;
195     lineLength = fieldDim / 4;
196
197     lv_style_copy(&lineStyle, &lv_style_plain);
198     lineStyle.line.width = 6;
199     lineStyle.line.opa = LV_OPA_100;
200     lineStyle.line.color = LV_COLOR_MAKE(0, 255, 0);
201     lv_obj_set_style(line, &lineStyle);
202
203     /**
204      * Status Label
205      */
206     statusLabel = lv_label_create(container, NULL);
207     lv_style_copy(&textStyle, &lv_style_plain);

```

```

208     textStyle.text.color = LV_COLOR_WHITE;
209     textStyle.text.opa = LV_OPA_100;
210     lv_obj_set_style(statusLabel, &textStyle);
211     lv_label_set_text(statusLabel, "No Odom Data Provided");
212     lv_obj_align(statusLabel, container, LV_ALIGN_CENTER,
213                 -lv_obj_get_width(container) / 2 + (lv_obj_get_width(container) -
214                 ↪ fieldDim) / 2,
215                 0);
216
217     /**
218     * Reset Button
219     */
220     {
221         lv_obj_t * btn = lv_btn_create(container, NULL);
222         lv_obj_set_size(btn, 100, 40);
223         lv_obj_align(
224             btn, NULL, LV_ALIGN_IN_TOP_MID,
225             -lv_obj_get_width(container) / 2 + (lv_obj_get_width(container) - fieldDim)
226             ↪ / 2, 0);
227         lv_obj_set_free_ptr(btn, this);
228         lv_btn_set_action(btn, LV_BTN_ACTION_PR, resetAction);
229
230         /**
231         * Button Style
232         */
233         lv_style_copy(&resetRel, &lv_style_btn_tgl_rel);
234         resetRel.body.main_color = mainColor;
235         resetRel.body.grad_color = mainColor;
236         resetRel.body.border.color = LV_COLOR_WHITE;
237         resetRel.body.border.width = 2;
238         resetRel.body.border.opa = LV_OPA_100;
239         resetRel.body.radius = 2;
240         resetRel.text.color = LV_COLOR_WHITE;
241
242         lv_style_copy(&resetPr, &resetRel);
243         resetPr.body.main_color = LV_COLOR_WHITE;
244         resetPr.body.grad_color = LV_COLOR_WHITE;
245         resetPr.text.color = mainColor;
246
247         lv_btn_set_style(btn, LV_BTN_STYLE_REL, &resetRel);
248         lv_btn_set_style(btn, LV_BTN_STYLE_PR, &resetPr);
249
250         /**
251         * Reset Button Label
252         */
253         lv_obj_t * label = lv_label_create(btn, NULL);
254         lv_obj_set_style(label, &textStyle);
255         lv_label_set_text(label, "Reset");
256     }
257     printf("Made an OdomDebug\n");
258 }
259
260 OdomDebug::~OdomDebug() { lv_obj_del(container); }

```

```

260  /**
261   * Sets the function to be called when a tile is pressed
262   * @param callback a function that sets the odometry state
263   */
264  void OdomDebug::setStateCallback(std::function<void(state_t state)> callback)
265  {
266      stateFnc = callback;
267  }
268
269  /**
270   * Sets the function to be called when the reset button is pressed
271   * @param callback a function that resets the odometry and sensors
272   */
273  void OdomDebug::setResetCallback(std::function<void()> callback) { resetFnc = callback;
↵ }
274
275  /**
276   * Sets the position of the robot in QUnits and puts the sensor data on the
277   * display
278   * @param state robot state - x, y, theta
279   * @param sensors encoder information - left, right, middle (optional)
280   */
281  void OdomDebug::setData(state_t state, sensors_t sensors)
282  {
283
284      // position in court units
285      double c_x = state.x.convert(court);
286      double c_y = state.y.convert(court);
287      double c_theta = state.theta.convert(radian);
288
289      // place point on field
290      lv_obj_set_pos(led, (c_x * fieldDim) - lv_obj_get_width(led) / 2,
291                      (c_y * fieldDim) - lv_obj_get_height(led) / 2 - 1);
292
293      // move start and end of line
294      linePoints[0] = {(int16_t)((c_x * fieldDim)), (int16_t)((c_y * fieldDim) -
↵ (lineWidth / 2))};
295      double newY = lineLength * sin(c_theta);
296      double newX = lineLength * cos(c_theta);
297      linePoints[1] = {(int16_t)(newX + linePoints[0].x), (int16_t)(newY +
↵ linePoints[0].y)};
298
299      lv_line_set_points(line, linePoints.data(), linePoints.size());
300      lv_obj_invalidate(line);
301
302      std::string text =
303          "X: " + std::to_string(std::round(state.x.convert(inch) * 1000) /
↵ 1000).substr(0, 5) +
304          " in\n" +
305          "Y: " + std::to_string(std::round(state.y.convert(inch) * 1000) /
↵ 1000).substr(0, 5) +
306          " in\n" + "Theta: " +
307          std::to_string(std::round(state.theta.convert(degree) * 1000) / 1000).substr(0,
↵ 5) +

```

```

308         " deg\n" + "Left: " + std::to_string(sensors.left).substr(0, 4) + " deg\n" +
309         "Right: " + std::to_string(sensors.right).substr(0, 4) + " deg\n";
310     if (sensors.hasMiddle)
311     {
312         text = text + "Middle: " + std::to_string(sensors.middle).substr(0, 4) + " deg";
313     }
314
315     lv_label_set_text(statusLabel, text.c_str());
316     lv_obj_align(statusLabel, container, LV_ALIGN_CENTER,
317                 -lv_obj_get_width(container) / 2 + (lv_obj_get_width(container) -
318                 ↪ fieldDim) / 2,
319                 25);
320 }
321 /**
322  * Sets odom state when tile is pressed
323  * Decodes tile ID to find position
324  */
325 lv_res_t OdomDebug::tileAction(lv_obj_t * tileObj)
326 {
327     OdomDebug * that = static_cast<OdomDebug *>(lv_obj_get_free_ptr(tileObj));
328     int num = lv_obj_get_free_num(tileObj);
329     int y = num / 6;
330     int x = num - y * 6;
331     if (that->stateFnc)
332         that->stateFnc({x * tile + 0.5_tl, y * tile + 0.5_tl, 0_deg});
333     else
334         std::cout << "OdomDebug: No tile action callback provided";
335     return LV_RES_OK;
336 }
337
338 /**
339  * Reset Sensors and Position
340  */
341 lv_res_t OdomDebug::resetAction(lv_obj_t * btn)
342 {
343     OdomDebug * that = static_cast<OdomDebug *>(lv_obj_get_free_ptr(btn));
344     if (that->resetFnc)
345         that->resetFnc();
346     else
347         std::cout << "OdomDebug: No reset action callback provided";
348     return LV_RES_OK;
349 }

```


3.6 src/movement/Drivetrain.cpp

```
1  /**
2   * Drivetrain.cpp
3   *
4   * This file contains the definitions of the Drivetrain class.
5   * The Drivetrain class handles almost everything relating to the
6   * drivetrain: motor control, settings (like max speed), basic
7   * movement methods (like tank or arcade), more advanced movement
8   * methods (like PID to point, path following, and motion
9   * profiling), and more.
10  */
11  #include "main.h" // gives access to Drivetrain and other dependencies
12
13  /* ----- Private Information ----- */
14  /* ----- Private Information ----- */
15  /* ----- Private Information ----- */
16
17  /* ----- Motor References ----- */
18  Motor & Drivetrain::mmtrLeftFront = def::mtr_dt_left_front;
19  Motor & Drivetrain::mmtrRightFront = def::mtr_dt_right_front;
20  Motor & Drivetrain::mmtrRightBack = def::mtr_dt_right_back;
21  Motor & Drivetrain::mmtrLeftBack = def::mtr_dt_left_back;
22
23  /* ----- Okapi Chassis ----- */
24  std::shared_ptr<ChassisController> Drivetrain::mchassis =
25      ChassisControllerBuilder()
26          .withMotors({Drivetrain::mmtrLeftFront, Drivetrain::mmtrLeftBack},
27                     {Drivetrain::mmtrRightFront, Drivetrain::mmtrRightBack})
28          .withDimensions(AbstractMotor::gearset::green,
29                          {{def::DRIVE_WHEEL_DIAMETER, def::DRIVE_OFFSET}, imev5GreenTPR})
30          .build(); // chassis object for using Pathfinder through okapi
31
32  /* ----- Protected Information ----- */
33  /* ----- Protected Information ----- */
34  /* ----- Protected Information ----- */
35
36  /* ----- Initial Settings ----- */
37  double Drivetrain::mmaxSpeed = def::SET_DT_MAX;
38  bool Drivetrain::menabled = true;
39
40  /* ----- Simple Follow Data ----- */
41  double Drivetrain::mlastLookIndex = 0; // index of the last lookahead point
42  double Drivetrain::mlastPartialIndex =
43      0; // fractional index of where the last lookahead point was on the segment
44
45  /* ----- Odometry Accessors ----- */
46  OdomState Drivetrain::getState() // get position as OdomState
47  {
48      return CustomOdometry::getState();
49  }
50  QLength Drivetrain::getXPos() { return CustomOdometry::getX(); }
51  QLength Drivetrain::getYPos() { return CustomOdometry::getY(); }
52  QAngle Drivetrain::getTheta() { return CustomOdometry::getTheta(); }
```

```

53 ExtendedPoint Drivetrain::getPoint() // get position as ExtendedPoint
54 {
55     return ExtendedPoint(getXPos(), getYPos(), getTheta());
56 }
57
58 /* ----- Helpers ----- */
59 QAngle Drivetrain::angleToPoint(
60     const Point & itargetPoint) // calculates the field centric direction to the
61     ↪ itargetPoint from
62     // the robot's current position
63 {
64     return (atan((getYPos().convert(inch) - itargetPoint.y.convert(inch)) /
65         (getXPos().convert(inch) - itargetPoint.x.convert(inch))) +
66         (getXPos() > itargetPoint.x ? M_PI : 0)) *
67         radian;
68 }
69 std::optional<double> Drivetrain::findIntersection(
70     ExtendedPoint istart, ExtendedPoint iend,
71     const double & ilookDistIn) // looks for interections between the line segment
72     ↪ created by the
73     // two points (istart and iend), and the circle around
74     ↪ the robot
75     // with radius ilookDistIn (lookahead circle)
76 {
77     ExtendedPoint d = iend - istart; // differece vector
78     ExtendedPoint f = istart - getPoint(); // robot position relative to the start of
79     ↪ the segment
80
81     double a = d.dot(d).convert(inch); // set up quadratic
82     double b = 2 * f.dot(d).convert(inch);
83     double c = f.dot(f).convert(inch) - ilookDistIn * ilookDistIn;
84
85     double discriminant = b * b - 4 * a * c; // used to make sure it doesn't sqrt(a
86     ↪ negative number)
87     if (discriminant >= 0)
88     {
89         discriminant = sqrt(discriminant);
90
91         double t1 = (-b - discriminant) / (2 * a); // solution 1
92         double t2 = (-b + discriminant) / (2 * a); // solution 2
93
94         if (t2 >= 0 && t2 <= 1) // t2 is always farther along the segment, so return t2
95             ↪ first
96         {
97             return t2;
98         }
99         else if (t1 >= 0 && t1 <= 1) // then t1
100         {
101             return t1;
102         }
103     }
104
105     return {}; // no intersections
106 }

```

```

101 ExtendedPoint Drivetrain::findLookahead(
102     SimplePath ipath, const double & ilookDistIn) // looks for the intersection point
103     ↪ between the                               // lookahead circle and the
104                                                     ↪ SimplePath, ipath
105 {
106     ExtendedPoint currentPos = getPoint();
107     int lastIntersectIndex = 0;
108
109     if (currentPos.dist(ipath.last()).convert(inch) <=
110         ilookDistIn) // if the last point is within range, return
111     {
112         return ipath.last();
113     }
114
115     for (int i = mlastLookIndex; i < ipath.size() - 1;
116         i++) // searches through the whole path starting at the index of the previous
117         ↪ lookahead point
118     {
119         std::optional<double> t_partialIndex = findIntersection(
120             ipath.at(i), ipath.at(i + 1),
121             ilookDistIn); // finds the partial index of the intersection in the range
122             ↪ [0, 1)
123
124         if (t_partialIndex.has_value() &&
125             (i > mlastLookIndex ||
126              t_partialIndex > mlastPartialIndex)) // if there is an intersection
127             ↪ farther along the
128                                                     // path than the last point
129         {
130             mlastLookIndex = i;
131             mlastPartialIndex = t_partialIndex.value();
132
133             if (lastIntersectIndex > 0) // if this is the second intersection, the loop
134             ↪ can exit
135             {
136                 break;
137             }
138
139             lastIntersectIndex = i; // if this is the first intersection found, record
140             ↪ it
141         }
142
143         if (lastIntersectIndex > 0 &&
144             ipath.at(i).dist(ipath.at(lastIntersectIndex)).convert(inch) >=
145                 ilookDistIn *
146                 2) // if it is searching for intersections farther than the
147                 ↪ diameter of a
148                     // lookahead circle, and it has already found a point, exit the
149                     ↪ loop. It is
150                     // impossible for there to be a second lookahead point more than
151                     ↪ 2 *
152                     // (lookahead distance) away from the first lookahead point
153         {
154             break;
155         }
156     }
157 }

```

```

145     }
146 }
147
148 ExtendedPoint segmentStart = ipath.at(mlastLookIndex);
149 def::display.setMiscData(
150     1, "start: " + segmentStart.string() +
151         "\nvec: " + (ipath.at(mlastLookIndex + 1) - segmentStart).string() +
152         "\npartlIndx: " + std::to_string(mlastPartialIndex) + "\nscaled: " +
153         ((ipath.at(mlastLookIndex + 1) - segmentStart) *
154             ↪ mlastPartialIndex).string());
154 return segmentStart +
155     (ipath.at(mlastLookIndex + 1) - segmentStart) *
156     mlastPartialIndex; // calculates the location of the lookahead point by
157         ↪ getting the
158         // vector from the start of the segment to the end of
159         ↪ the segment,
160         // multiplying that by the fractional index of the
161         ↪ lookahead
162         // point, and adding that the the starting point of
163         ↪ the segment
164 }
165
166 /* ----- Public Information ----- */
167 /*                                     */
168 /* ----- */
169 std::shared_ptr<AsyncMotionProfileController> Drivetrain::mprofiler =
170     AsyncMotionProfileControllerBuilder()
171         .withLimits({def::DRIVE_MAX_SPEED.convert(mps),
172             ↪ def::DRIVE_MAX_ACCEL.convert(mps2), 8.0})
173         .withOutput(Drivetrain::mchassis)
174         .buildMotionProfileController(); // okapi motion profile controller with
175             ↪ measured constants
176
177 /* ----- Getters/Setters ----- */
178 double Drivetrain::getMaxSpeed() { return mmaxSpeed; }
179 void Drivetrain::setMaxSpeed(double imaxSpeed) { mmaxSpeed = imaxSpeed; }
180
181 bool Drivetrain::isEnabled() { return menabled; }
182 void Drivetrain::enable() // allows movements to be startable
183 {
184     menabled = true;
185 }
186 void Drivetrain::disable() // stops active movemnts
187 {
188     menabled = false;
189     moveTank(0, 0, false);
190 }
191
192 void Drivetrain::checkNextAsync(
193     const double & ierror,
194     std::vector<AsyncAction> & iactions) // checks if the next AsyncAction should
195         ↪ execute, and
196         // executes it (and removes it from the list)
197         ↪ if it should

```

```

190 {
191     if (iactions.size()) // if there is at least one action to execute
192     {
193         const AsyncAction & nextAction = iactions.at(0);
194         if (ierror < nextAction.merror) // if the robot is close enough to the target
195         {
196             nextAction.maction(); // execute the action
197             iactions.erase(iactions.begin()); // remove the action, having already
198                 ↪ executed it
199         }
200     }
201 }
202
203 /* ----- Basic Movement ----- */
204 * These "basic" motions are lower level fuctions mostly just intended
205 * to prevent the call to each motor individually in more advanced
206 * motions, to keep the code cleaner.
207 * "Saturation" is when the motor inputs are higher than their max
208 * speed, which makes the motor go at max speed. This can cause problems,
209 * however, when the motors are all working together to follow a
210 * specific motion, because one motor might be going at exactly the
211 * intended speed, but another motor might be saturated, so it doesn't
212 * go at the right speed, making the robot follow the wrong motion.
213 * To account for this, the basic movment methods have a variable,
214 * idesaturate, that, when true, scales all motor values down so they
215 * fit within the motors capabilities.
216 */
217 void Drivetrain::moveIndependant(
218     double ileftFront, double  irightFront, double  irightBack, double  ileftBack,
219     const bool idesaturate) // moves each motor {lf, rf, rb, lb} in range [-1,1]
220 {
221     if (idesaturate) // desaturates values
222     {
223         std::array<double, 4> motor_values =
224             util::scaleToFit<4>(mmaxSpeed, {ileftFront, irightFront, irightBack,
225                 ↪ ileftBack});
226         ileftFront = motor_values[0];
227         irightFront = motor_values[1];
228         irightBack = motor_values[2];
229         ileftBack = motor_values[3];
230     }
231     // moves all of the motors by voltage
232     mmtrLeftFront.moveVoltage(12000 * ileftFront);
233     mmtrRightFront.moveVoltage(12000 * irightFront);
234     mmtrRightBack.moveVoltage(12000 * irightBack);
235     mmtrLeftBack.moveVoltage(12000 * ileftBack);
236 }
237 void Drivetrain::moveTank(double ileft, double  iright,
238     const bool idesaturate) // spins the left side and right side
239     ↪ motors at
240     // certian speeds in range [-1,1]
241 {
242     if (idesaturate) // desaturates values

```

```

241 {
242     std::array<double, 2> motor_values = util::scaleToFit<2>(mmaxSpeed, {ileft,
        ↳ iright});
243     ileft = motor_values[0];
244     iright = motor_values[1];
245 }
246 Drivetrain::moveIndependant(
247     ileft, iright, iright, ileft,
248     false); // don't try to desaturate, because the values have already been
        ↳ desaturated
249 }
250 void Drivetrain::moveArcade(
251     double iforward, double istrafe, double iturn,
252     const bool idesaturate) // moves the robot with arcade-style inputs in range [-1,1]
253 {
254     if (idesaturate) // desaturates values
255     {
256         std::array<double, 4> motor_values = {
257             iforward + istrafe + iturn, iforward - istrafe - iturn, iforward + istrafe
                ↳ - iturn,
258             iforward - istrafe + iturn};
259         util::scaleToFit<4>(mmaxSpeed, motor_values); // modifies reference to
            ↳ motor_values
260         Drivetrain::moveIndependant(motor_values[0], motor_values[1], motor_values[2],
261             motor_values[3]); // moves the motors from within
                ↳ the if to
262                                     // prevent the need to copy values
263     }
264     else
265     {
266         Drivetrain::moveIndependant(iforward + istrafe + iturn, iforward - istrafe -
            ↳ iturn,
267             iforward + istrafe - iturn, iforward - istrafe +
                ↳ iturn,
268             false); // don't desaturate
269     }
270 }
271
272 /* ----- Intermediate Movement ----- */
273 void Drivetrain::moveInDirection(
274     QAngle idirection, const bool ifieldCentric, double imagnitude, double itheta,
275     const bool idesaturate) // moves the robot with a certain speed in a certain
        ↳ direction, while
276                                     // turning a certain amount
277 {
278     if (ifieldCentric) // if the direction is in reference to the field
279     {
280         idirection -= Drivetrain::getTheta(); // changes the direction the robot should
            ↳ go in based
281                                     // on its field centric rotation
282     }
283     idirection = util::wrapQAngle(idirection); // fits idirection into [0, 360)
284     util::chop<double>(0, 1, imagnitude); // caps magnitude
285     util::chop<double>(-1, 1, itheta); // caps itheta

```

```

286
287     Drivetrain::moveArcade(
288         imagitude * cos(idirection.convert(radian)), imagitude *
289         ↪ sin(idirection.convert(radian)),
290         itheta, idesaturate); // move in the direction of the vector, and turn the
291         ↪ specified amount
292     }
293
294     /* ----- Move to Point Methods ----- */
295     * Because these methods have a target, they need to be run in a loop
296     * to constantly re-evaluate how fast the robot should be going. To
297     * do this, there are PID and Slew calculations being done for driving
298     * straight and turning in each of the methods. This is easy to do
299     * because of the PID and Slew classes.
300     *
301     * Each method has custom tuned default PID/Slew values, but they
302     * can be modified on a per-motion basis when they are called.
303     */
304     void Drivetrain::strafeToPoint(
305         ExtendedPoint itarget, std::vector<AsyncAction> iactions, PID imagitudePID, PID
306         ↪ iturnPID,
307         Slew imagitudeSlew,
308         Slew iturnSlew) // drives in a stright line to the point while turning using set
309         ↪ PID/Slew gains,
310         ↪ // and executing the AsyncActions at the right times
311     {
312         enable(); // make sure the action can run
313         while (menabled && (!imagitudePID.isSettled() || !iturnPID.isSettled()))
314         {
315             double inToPoint =
316                 OdomMath::computeDistanceToPoint(itarget, Drivetrain::getState())
317                 .convert(inch); // calc inches to target point. itarget can be passed as
318                 ↪ // okapi::Point, because ExtendedPoint inherits from
319                 ↪ okapi::Point
320
321             double degToPoint = util::wrapDeg180(
322                 (itarget.theta - Drivetrain::getTheta())
323                 .convert(degree)); // calc the angle to the point in the range [-180,
324                 ↪ 180) to always
325                 ↪ // turn the right direction
326             def::display.setMiscData(1, std::to_string(degToPoint));
327
328             Drivetrain::moveInDirection(Drivetrain::angleToPoint(itarget), true,
329                 imagitudeSlew.iterate(imagitudePID.iterate(inToPoint)
330                 ↪ int)),
331                 iturnSlew.iterate(iturnPID.iterate(degToPoint)),
332                 ↪ true);
333
334             Drivetrain::checkNextAsync(
335                 inToPoint,
336                 iactions); // executes the next action if available, and removes it from
337                 ↪ the list
338
339             pros::delay(20);
340         }
341     }

```

```

331 }
332
333 void Drivetrain::straightToPoint(
334     ExtendedPoint itarget, std::vector<AsyncAction> iactions, QLength inoTurnRange,
335     double iturnWeight, PID imagnitudePID, PID iturnPID, Slew imagnitudeSlew,
336     Slew iturnSlew) // drives to the point without strafing using set PID/Slew gains,
    ↪ and executing
    ↪ // the AsyncActions at the right times
337 {
338     const double noTurnRangeIn = inoTurnRange.convert(inch);
339
340     enable(); // make sure the action can run
341     while (menabled && !imagnitudePID.isSettled())
342     {
343         QAngle angleToPoint = util::wrapQAngle180(
344             Drivetrain::angleToPoint(itarget) -
345             Drivetrain::getTheta()); // how much the robot needs to turn to face the
    ↪ point
346         double inToPoint = OdomMath::computeDistanceToPoint(itarget,
    ↪ Drivetrain::getState())
    ↪ .convert(inch); // how far the robot is away from the
    ↪ target
347         double inForward =
348             inToPoint *
349             cos(angleToPoint.convert(radian)); // how far the robot needs to drive
    ↪ straight (no
    ↪ // turning) to get as close to the
    ↪ target as possible
350
351         double forward = imagnitudeSlew.iterate(
352             imagnitudePID.iterate(inForward)); // calculates value from PID fed into
    ↪ Slew
353         util::chop<double>(-mmaxSpeed, mmaxSpeed,
354             forward); // limits the values in [-mmaxSpeed, mmaxSpeed]
355
356         double turn;
357         if (inToPoint > noTurnRangeIn) // if the robot is far enough away from the
    ↪ target
358         {
359             turn = iturnSlew.iterate(iturnPID.iterate(
360                 angleToPoint.convert(degree))); // calculates value from PID fed into
    ↪ Slew
361             util::chop<double>(-mmaxSpeed, mmaxSpeed, turn);
362         }
363         else
364         {
365             turn = 0; // don't turn when too close to the target
366         }
367
368         if (abs(turn) == mmaxSpeed) // if the robot is turning at max speed (which
    ↪ means it must be
    ↪ // far off target)
369         {
370
371
372
373

```



```

374         turn *= iturnWeight; // increase the amount to turn, so that it turns
           ↳ faster as a result
375                                     // of forward getting scaled down in moveArcade
376     }
377
378     Drivetrain::checkNextAsync(
379         inToPoint,
380         iactions); // executes the next action if available, and removes it from
           ↳ the list
381
382     Drivetrain::moveArcade(forward, 0, turn, true);
383
384     pros::delay(20);
385 }
386 }
387
388 void Drivetrain::arcStraightToPoint(
389     ExtendedPoint itarget, std::vector<AsyncAction> iactions, double iweightModifier,
390     QLength inoTurnRange, PID imagnitudePID, PID iturnPID, Slew imagnitudeSlew,
391     Slew iturnSlew) // drive in an "arc" (doesn't follow a path, just approximates an
           ↳ arc) using set
392                                     // PID/Slew gains, and executing the AsyncActions at the right times
393 {
394     const double noTurnRangeIn = inoTurnRange.convert(inch);
395
396     enable(); // make sure the action can run
397     while (menabled && !imagnitudePID.isSettled())
398     {
399         double theta = util::wrapRad(
400             2 * (Drivetrain::getTheta().convert(radian) -
401                 abs(atan2(
402                     (Drivetrain::getYPos() - itarget.y).convert(inch),
403                     (Drivetrain::getXPos() - itarget.x)
404                         .convert(inch))))); // calculates how much the robot should
           ↳ end up turning
405         double radius = abs(
406             hypot((getXPos() - itarget.x).convert(inch), (getYPos() -
407                 ↳ itarget.y).convert(inch)) /
408             2 / sin(theta / 2)); // calculates the radius of the arc
409         double targetIn = theta * radius; // how far the robot needs to go (arc length)
410         double turnWeight =
411             iweightModifier /
412             radius; // how aggressively the robot needs to turn to approximate the arc
413
414         QAngle angleToPoint =
415             Drivetrain::angleToPoint(itarget) - Drivetrain::getTheta(); // direction of
416             ↳ target
417         double inToPoint = OdomMath::computeDistanceToPoint(itarget, getState())
418             .convert(inch); // distance of target
419         double inForward =
420             inToPoint *
421             cos((angleToPoint)
422                 .convert(radian)); // distance to perpendicular line intersecting
           ↳ target

```

```

421
422     double forward = imagnitudeSlew.iterate(
423         imagnitudePID.iterate(inForward)); // calculates value from PID fed into
         ↳ Slew
424     util::chop<double>(-mmaxSpeed, mmaxSpeed,
425         forward); // limits the values in [-mmaxSpeed, mmaxSpeed]
426
427     double turn;
428     if (inToPoint > noTurnRangeIn) // if the robot is far enough away from the
         ↳ target
429     {
430         turn = iturnSlew.iterate(iturnPID.iterate(
431             angleToPoint.convert(degree))); // calculates value from PID fed into
             ↳ Slew
432         util::chop<double>(-mmaxSpeed, mmaxSpeed, turn);
433     }
434     else
435     {
436         turn = 0; // don't turn when too close to the target
437     }
438
439     if (abs(turn) == mmaxSpeed) // if the robot is turning at max speed (which
         ↳ means it must be
440                                     // far off target)
441     {
442         turn *= turnWeight; // increase the amount to turn, so that it turns faster
         ↳ as a result
443                                     // of forward getting scaled down in moveArcade
444     }
445
446     Drivetrain::checkNextAsync(
447         inToPoint,
448         iactions); // executes the next action if available, and removes it from
         ↳ the list
449
450     Drivetrain::moveArcade(forward, 0, turn, true);
451
452     pros::delay(20);
453 }
454 }
455
456 /* ----- Path Following Methods ----- /
457 * simpleFollow uses the concept from "Pure Pursuit" of using a "lookahead circle" to
   ↳ follow a path.
458 * The idea is that, when given a line, the robot will figure out how to follow it
   ↳ smoothly. It does
459 * this by checking for points on the line that are a certain distance away from the
   ↳ robot, and
460 * moving in the direction of whichever point it sees that is farthest on the line.
   ↳ Another way to
461 * picture this, is the robot has a circle (the lookahead circle) around it with the
   ↳ radius being
462 * the "lookahead distance". The robot is always trying to drive to intersections
   ↳ between this

```

```

463  * circle, and the path (a.k.a. the lookahead point).
464  *
465  * The robot goes at full speed to the lookahead point until it gets to the end, where
↳ it settles
466  * with PID.
467  */
468 void Drivetrain::simpleFollow(
469     SimplePath ipath, QLength ilookDist, std::vector<AsyncAction> iactions, PID
↳ imagnitudePID,
470     PID iturnPID, Slew imagnitudeSlew,
471     Slew iturnSlew) // follows the path, ipath using set lookahead distance (ilookDist)
↳ and PID/Slew
472                     // gains while executing the AsyncActions at the right times (only
↳ on the last
473                     // segment)
474 {
475     double lookDistIn = ilookDist.convert(inch);
476     ExtendedPoint lookPoint = ipath.at(0);
477
478     double magnitude = 1; // the robot will always go full speed until the end is near
479     bool reachedEnd = false;
480
481     mlastLookIndex = 0;
482     mlastPartialIndex = 0;
483
484     enable(); // make sure the action can run
485     while (menabled && (!imagnitudePID.isSettled() || !iturnPID.isSettled()) ||
↳ !reachedEnd)
486     {
487         if (!reachedEnd &&
488             mlastLookIndex ==
489             ipath.size() - 2) // detects if the robot should be going to the last
↳ point
490         {
491             reachedEnd = true;
492             lookPoint = ipath.last();
493         }
494
495         if (!reachedEnd)
496         {
497             lookPoint = findLookahead(ipath, lookDistIn); // find the next lookahead
498         }
499         else
500         {
501             double inToPoint =
502                 OdomMath::computeDistanceToPoint(lookPoint, Drivetrain::getState())
503                 .convert(
504                     inch); // calc inches to target point. itarget can be passed as
505                             // okapi::Point, because ExtendedPoint inherits from
↳ okapi::Point
506
507             Drivetrain::checkNextAsync(
508                 inToPoint,

```

```

509         iactions); // executes the next action if available, and removes it
                    ↪ from the list
510
511         magnitude = imagitudePID.iterate(inToPoint); // how fast the robot should
                    ↪ be moving
512     }
513
514     double degToPoint = util::wrapDeg180(
515         (lookPoint.theta - Drivetrain::getTheta())
516         .convert(degree)); // calc the angle to the point in the range [-180,
                    ↪ 180) to always
517                             // turn the right direction
518
519     moveInDirection(Drivetrain::angleToPoint(lookPoint), true,
520                     imagitudeSlew.iterate(magnitude),
521                     iturnSlew.iterate(iturnPID.iterate(degToPoint)), true);
522
523     pros::delay(20);
524 }
525 }
526
527 /* ----- Motion Profiling ----- */
528 void Drivetrain::generatePathToPoint(
529     PathfinderPoint ipoint,
530     const std::string & iname) // use Pathfinder through okapi to make a motion profile
531 {
532     ipoint.y = -ipoint.y;
533     ipoint.theta = -ipoint.theta;
534     mprofiler->generatePath({0_ft, 0_ft, 0_deg}, ipoint, iname);
535 }
536 void Drivetrain::followPathfinder(const std::string & iname, bool ibackwards,
537                                   bool imirrored) // follow Pathfinder path through
                    ↪ okapi
538 {
539     mprofiler->setTarget(iname, ibackwards, imirrored);
540 }
541 void Drivetrain::followTraj(Trajectory & itraj) // follow trajectory loaded from sd card
542 {
543     const double startLeft = mmtrLeftFront.getPosition();
544     const double startRight = mmtrRightFront.getPosition();
545
546     if (itraj.isDone()) // if the path is done before execution, reset
547     {
548         itraj.reset();
549     }
550     while (!itraj.isDone()) // execute until the path is done
551     {
552         std::pair<double, double> values = itraj.iterate(
553             (mmtrLeftFront.getPosition() - startLeft) *
554             ↪ def::DRIVE_WHEEL_CIRCUMFERENCE_IN / 360,
555             (mmtrRightFront.getPosition() - startRight) *
556             ↪ def::DRIVE_WHEEL_CIRCUMFERENCE_IN /
557             360); // iterate through the profile passing the distance each side has
                    ↪ gone so far

```

```
556 |  
557     moveTank(values.first, values.second, false);  
558  
559     pros::delay(10);  
560 }  
561 }
```

3.7 src/movement/paths/ProfileStep.cpp

```
1  /**
2   * ProfileStep.cpp
3   *
4   * ProfileStep is used for organizing the information parsed from motion profiles
   ↳ stored on the sd
5   * card, calculated by the publically available GitHub repository, TrajectoryLib by
   ↳ Team254 (FRC
6   * Team 254, The Cheesy Poofs), found here: https://github.com/Team254/TrajectoryLib.
   ↳ The
7   * trajectories are calculated on a computer, and stored on the sd card for the robot
   ↳ to use. Each
8   * time step of the profile is read from the sd card, and stored in an instance of
   ↳ ProfileStep by
9   * the Trajectory class.
10  */
11  #include "main.h" // gives access to ProfileStep declaration and other dependencies
12
13  const std::string ProfileStep::getString() // returns the ProfileStep formatted as a
   ↳ std::string
14                                          // without changing anything (const)
15  {
16      return std::to_string(pos) + " " + std::to_string(vel) + " " + std::to_string(acc)
   ↳ + " " +
17          std::to_string(jerk) + " " + std::to_string(heading) + " " +
   ↳ std::to_string(dt) + " " +
18          std::to_string(x) + " " + std::to_string(y);
19  }
```

3.8 src/movement/paths/SimplePath.cpp

```
1  /**
2   * SimplePath.cpp
3   *
4   * SimplePath is a simple struct that has a list of points
5   * on a path represented by ExtendedPoints in a std::vector.
6   * This is used for path following by the Drivetrain class.
7   */
8  #include "main.h" // gives access to SimplePath and other dependencies
9
10 ExtendedPoint SimplePath::at(size_t iindex) // returns the point at the index
11 {
12     return mpoints.at(iindex);
13 }
14 ExtendedPoint SimplePath::last() // returns the point at the end
15 {
16     return mpoints.back();
17 }
18 int SimplePath::size() // returns the length of the path
19 {
20     return mpoints.size();
21 }
```

3.9 src/movement/paths/Trajectory.cpp

```
1  /**
2   * Trajectory.cpp
3   *
4   * This file contains the definitions of the Trajectory class. The Trajectory class
5   * ↪ reads and stores
6   * motion profile information from the sd card. Motion profiles stored on the sd card
7   * ↪ are calculated
8   * by the publically available GitHub repository, TrajectoryLib by Team254 (FRC Team
9   * ↪ 254, The Cheesy
10  * Poofs), found here: https://github.com/Team254/TrajectoryLib. The trajectories are
11  * ↪ calculated on
12  * a computer, and stored on the sd card for the robot to use. Each time step of the
13  * ↪ profile is read
14  * from the sd card, and stored in an instance of ProfileStep by the Trajectory class.
15  *
16  * The paths are intended to be executed by the Drivetrain class, but are not used in
17  * ↪ programming
18  * skills.
19  */
20 #include "main.h" // gives access to Trajectory and other dependencies
21
22 /* ----- */
23 /*                               Public Information                               */
24 /* ----- */
25
26 Trajectory::Trajectory(const char * ifileName, double ikP, double ikD, double ikV,
27 ↪ double ikA)
28 : mkP(ikP), mkD(ikD), mkV(ikV), mkA(ikA), mstepNumber(0), mlastErrorL(0.0),
29   mlastErrorR(0.0) // constructor that specifies the file with the trajectory, and
30 ↪ the gains for
31 ↪ following the trajectory
32 {
33     FILE * file; // creates a file object to be used later
34     if (pros::usd::is_installed()) // checks if the sd card is installed before trying
35     ↪ to access it
36     {
37         file = fopen(ifileName, "r"); // open the file
38         if (file) // makes sure the file was opened correctly
39         {
40             char name[256];
41             fgets(name, 255, file); // put the name of the trajectory in a char array
42             mname = name;
43             mname = mname.substr(0, mname.length() - 1); // chop off \n
44
45             fscanf(file, "%i", &mlength); // store the number of steps
46
47             mleftSteps = new ProfileStep[mlength]; // dynamically allocate left and
48             ↪ right profiles
49
50             // based on the length of the
51             ↪ trajectory
52             mrightSteps = new ProfileStep[mlength];
53
54             for (int i = 0; i < mlength; i++) // fill left profile array from sd card
```



```

42     {
43         float pos, vel, acc, jerk, heading, dt, x, y;
44         fscanf(file, "%f %f %f %f %f %f %f %f", &pos, &vel, &acc, &jerk,
45             ↪ &heading, &dt, &x,
46             &y);
47         mleftSteps[i] = {pos, vel, acc, jerk, heading, dt, x, y};
48     }
49     for (int i = 0; i < mlength; i++) // fill right profile array from sd card
50     {
51         float pos, vel, acc, jerk, heading, dt, x, y;
52         fscanf(file, "%f %f %f %f %f %f %f %f", &pos, &vel, &acc, &jerk,
53             ↪ &heading, &dt, &x,
54             &y);
55         mrightSteps[i] = {pos, vel, acc, jerk, heading, dt, x, y};
56     }
57 }
58 else
59 {
60     std::cout << "\"" << ifileName << "\"" file is null"
61         << std::endl; // output to the terminal if the file is null
62 }
63
64 fclose(file);
65 }
66 else // if the sd card is not installed, create empty arrays and send a message to
67     ↪ the terminal
68 {
69     mleftSteps = new ProfileStep[1];
70     mrightSteps = new ProfileStep[1];
71     std::cout << "no sd card inserted" << std::endl;
72 }
73 }
74 Trajectory::~Trajectory() // destructor to clean up heap variables
75 {
76     delete[] mleftSteps;
77     delete[] mrightSteps;
78 }
79
80 int Trajectory::getLength() { return mlength; }
81 std::string Trajectory::getName() { return mname; }
82 void Trajectory::reset() { mstepNumber = 0; }
83 std::pair<ProfileStep, ProfileStep>
84 Trajectory::getStep(int istepNumber) // return the left and right ProfileSteps at a
85     ↪ given point
86 {
87     if (istepNumber >
88         sizeof(mleftSteps) / sizeof(ProfileStep *)) // if the stepnumber is out of range
89     {
90         std::cout << "index out of bounds" << std::endl;
91     }
92     return std::pair<ProfileStep, ProfileStep>(mleftSteps[istepNumber],
93         ↪ mrightSteps[istepNumber]);
94 }

```

```

90 void Trajectory::setGains(const double ikP, const double ikD, const double ikV, const
    ↪ double ikA)
91 {
92     mkP = ikP;
93     mkD = ikD;
94     mkV = ikV;
95     mkA = ikA;
96 }
97 bool Trajectory::isDone() // checks to see if the trajectory is done
98 {
99     return mlength <= mstepNumber;
100 }
101
102 std::pair<double, double>
103 Trajectory::iterate(double ileftDistSoFar,
104                     double  irightDistSoFar) // goes through one iteration of the
    ↪ FEEDFORWARD loop
105                                     // based on how far the left and right
    ↪ wheels have gone
106 {
107     if (mstepNumber < mlength)
108     {
109         double errorL = mleftSteps[mstepNumber].pos -
110             ileftDistSoFar; // difference between where the wheels should
    ↪ be, and where
111                                     // they are for feedback control
112         double errorR = mrightSteps[mstepNumber].pos - irightDistSoFar;
113         double errorVelL = ((errorL - mlastErrorL) / mleftSteps[mstepNumber].dt -
114             mleftSteps[mstepNumber].vel); // velocity error
115         double errorVelR =
116             ((errorR - mlastErrorR) / mrightSteps[mstepNumber].dt -
    ↪ mrightSteps[mstepNumber].vel);
117
118         double resultL =
119             mkP * errorL + mkD * errorVelL + mkV * mleftSteps[mstepNumber].vel +
120             mkA * mleftSteps[mstepNumber].acc; //  $K_p * e_p(t) + K_d * e_v(t) + K_v * r_v(t) +$ 
    ↪  $K_a * r_a(t)$ 
121         double resultR = mkP * errorR + mkD * errorVelR + mkV *
    ↪ mrightSteps[mstepNumber].vel +
122             mkA * mrightSteps[mstepNumber].acc;
123
124         mlastErrorL = errorL; // store error for derivative calculation next time
125         mlastErrorR = errorR;
126
127         mstepNumber++;
128
129         return {resultL, resultR}; // return the necessary motor movements
130     }
131     else
132     {
133         return {0, 0}; // return 0 power for both motors, because the path has finished
    ↪ executing
134     }
135 }

```

3.10 src/stateMachines/BallControlStateMachine.cpp

```
1  /**
2   * BallControlStateMachine.cpp
3   *
4   * This file contains the definitions of the BallControlStateMachine class.
5   * BallControlStateMachine inherits from VStateMachine, and
6   * it is responsible for controlling all of the ball manipulators
7   * (intake, indexer, filter, and flywheel).
8   *
9   * The intake, indexer, and flywheel all have their own mini
10  * state machine in structs all contained in
11  * BallControlStateMachine. BallControlStateMachine puts them
12  * all together to make them function cohesively
13  */
14  #include "main.h" // gives access to BallControlStateMachine and other dependencies
15
16  /* ----- */
17  /*                               Public Information                               */
18  /* ----- */
19  BallControlStateMachine::BallControlStateMachine()
20      : controlEnabled(true), mbtnIn(def::btn_bc_in), mbtnOut(def::btn_bc_out),
21        mbtnShoot(def::btn_bc_shoot), mbtnFilter(def::btn_bc_down)
22  {
23  } // constructor to set defaults
24
25  void BallControlStateMachine::controlState() // sets the mini states based on inputs
26      ↪ from the                               // controller
27  {
28      if (mbtnOut.changedToPressed()) // when the out button is pressed
29      {
30          // spin out
31          mintake.mstate = IT_STATES::out;
32          mindexer.mstate = IX_STATES::out;
33      }
34      else if (mbtnOut.changedToReleased()) // when the out button is released
35      {
36          if (mbtnIn.isPressed()) // if the in button is also being pressed
37          {
38              // spin in
39              mintake.mstate = IT_STATES::in;
40              mindexer.mstate = IX_STATES::in;
41          }
42          else
43          {
44              // stop spinning
45              mintake.mstate = IT_STATES::off;
46              mindexer.mstate = IX_STATES::off;
47          }
48      }
49
50      if (mbtnIn.changedToPressed()) // if the in button is pressed
51      {
```

```

52         // spin in
53         mintake.mstate = IT_STATES::in;
54         mindexer.mstate = IX_STATES::in;
55     }
56     else if (mbtnIn.changedToReleased()) // if the in button is released
57     {
58         if (mbtnOut.isPressed()) // is the out button is also being pressed
59         {
60             // spin out
61             mintake.mstate = IT_STATES::out;
62             mindexer.mstate = IX_STATES::out;
63         }
64         else
65         {
66             // stop
67             mintake.mstate = IT_STATES::off;
68             mindexer.mstate = IX_STATES::off;
69         }
70     }
71
72     if (mbtnFilter.changedToPressed()) // if the filter button is pressed
73     {
74         mflywheel.mstate = FW_STATES::filter; // spin the filter
75     }
76     if (mbtnFilter.changedToReleased()) // is the filter button is released
77     {
78         if (mbtnShoot.isPressed()) // if the shoot button is also being pressed
79         {
80             mflywheel.mstate = FW_STATES::shoot; // shoot
81         }
82         else
83         {
84             mflywheel.mstate = FW_STATES::off; // stop spinning
85         }
86     }
87
88     if (mbtnShoot.changedToPressed()) // if the shoot button is pressed
89     {
90         mflywheel.mstate = FW_STATES::shoot; // shoot
91     }
92     else if (mbtnShoot.changedToReleased()) // if the shoot button is released
93     {
94         if (mbtnFilter.isPressed()) // if the filter button is also being pressed
95         {
96             mflywheel.mstate = FW_STATES::filter; // spin the filter
97         }
98         else
99         {
100             mflywheel.mstate = FW_STATES::off; // stop shooting
101         }
102     }
103 }
104 void BallControlStateMachine::update() // controls the robot based on the state by
    ↪ updating each

```

```

105                                     // mini state machine independantly
106 {
107     mintake.update();
108     mindexer.update();
109     mflywheel.update();
110 }
111
112 void BallControlStateMachine::itIn() // spins the intakes in
113 {
114     mintake.mstate = IT_STATES::in;
115 }
116 void BallControlStateMachine::itOut() // spins the intakes out
117 {
118     mintake.mstate = IT_STATES::out;
119 }
120 void BallControlStateMachine::itOff() // stops the intakes
121 {
122     mintake.mstate = IT_STATES::off;
123 }
124 void BallControlStateMachine::ixUp() // spins the indexer up
125 {
126     mindexer.mstate = IX_STATES::in;
127 }
128 void BallControlStateMachine::ixDown() // spins the indexer down
129 {
130     mindexer.mstate = IX_STATES::out;
131 }
132 void BallControlStateMachine::ixOff() // stops the indexer
133 {
134     mindexer.mstate = IX_STATES::off;
135 }
136 void BallControlStateMachine::fwShoot() // shoots the flywheel
137 {
138     mflywheel.mstate = FW_STATES::shoot;
139 }
140 void BallControlStateMachine::fwFilter() // spins the flywheel backwards
141 {
142     mflywheel.mstate = FW_STATES::filter;
143 }
144 void BallControlStateMachine::fwOff() // stops the flywheel
145 {
146     mflywheel.mstate = FW_STATES::off;
147 }
148
149 void BallControlStateMachine::itInFor(
150     double imillisecons) // spins the intakes for specified number of milliseconds
151 {
152     mintake.mstate = IT_STATES::in;
153     pros::delay(imillisecons);
154     mintake.mstate = IT_STATES::off;
155 }
156 void BallControlStateMachine::ixUpFor(
157     double imillisecons) // spins the indexer for specified number of milliseconds
158 {

```

```

159     mindexer.mstate = IX_STATES::in;
160     pros::delay(1000);
161     mindexer.mstate = IX_STATES::off;
162 }
163 void BallControlStateMachine::shoot(int ims) // shoots a ball
164 {
165     mflywheel.mstate = FW_STATES::shoot;
166     pros::delay(ims);
167     mflywheel.mstate = FW_STATES::off;
168 }
169 /* ----- */
170 /*           Private Information           */
171 /* ----- */
172
173 /* ----- Nested Classes ----- */
174 BallControlStateMachine::MIntake::MIntake()
175     : mstate(IT_STATES::off), mmotors({def::mtr_it_left, def::mtr_it_right})
176 {
177 } // constructor to set defaults
178 void BallControlStateMachine::MIntake::update() // updates the subsystem based on the
179     ↪ state
180 {
181     switch (mstate)
182     {
183     case IT_STATES::off:
184         mmotors.moveVoltage(0);
185         break;
186     case IT_STATES::in:
187         mmotors.moveVoltage(12000);
188         break;
189     case IT_STATES::out:
190         mmotors.moveVoltage(-12000);
191         break;
192     }
193 }
194 /* ----- */
195 BallControlStateMachine::MIndexer::MIndexer()
196     : mstate(IX_STATES::off), mmotor(def::mtr_ix) {} // constructor to set defaults
197 void BallControlStateMachine::MIndexer::update() // updates the subsystem based on the
198     ↪ state
199 {
200     switch (mstate)
201     {
202     case IX_STATES::off:
203         mmotor.moveVoltage(0);
204         break;
205     case IX_STATES::in:
206         mmotor.moveVoltage(12000);
207         break;
208     case IX_STATES::out:
209         mmotor.moveVoltage(-12000);
210         break;
211     }
212 }

```

```

211
212 BallControlStateMachine::MFlywheel::MFlywheel()
213     : mstate(FW_STATES::off), mmotors({def::mtr_fw1, def::mtr_fw2})
214 {
215 } // constructor to set defaults
216 void BallControlStateMachine::MFlywheel::update() // updates the subsystem based on the
    ↪ state
217 {
218     switch (mstate)
219     {
220         case FW_STATES::off:
221             mmotors.moveVoltage(0);
222             break;
223         case FW_STATES::shoot:
224             mmotors.moveVoltage(12000);
225             break;
226         case FW_STATES::filter:
227             mmotors.moveVoltage(-12000);
228             break;
229     }
230 }

```

3.11 src/stateMachines/DrivetrainStateMachine.cpp

```
1  /**
2   * DrivetrainStateMachine.cpp
3   *
4   * This file contains the definitions of the DrivetrainStateMachine class.
5   * DrivetrainStateMachine is a state machine that inherits from VStateMachine.
6   * It has an enumeration of different possible states to make it easy for
7   * the user to controll the drivetrain.
8   *
9   * To use the state machine in auton, you use doAutonMotion() to disable
10  * the normal state machine tasks and run the specified action.
11  */
12  #include "main.h" // gives access to DrivetrainStateMachine and other dependencies
13
14  /* ----- */
15  /*           Private Information           */
16  /* ----- */
17
18  /* ----- State ----- */
19  bool DrivetrainStateMachine::stateChanged() // returns whether the last state is the
20  ↪ same as the current one
21  {
22      if (mstate != mlastState)
23      {
24          return true;
25      }
26      return false;
27  }
28
29  /* ----- */
30  /*           Public Information           */
31  /* ----- */
32
33  DrivetrainStateMachine::DrivetrainStateMachine() : mstate(DT_STATES::off),
34  ↪ mlastState(mstate), mcontroller(def::controller),
35  ↪ mtoggleFieldCentric(def::btn_dt_tglFieldCentric), mdrivetrain(def::drivetrain) {}
36  ↪ // constructor to set defaults
37
38  DT_STATES DrivetrainStateMachine::getState() { return mstate; }
39  void DrivetrainStateMachine::setState(DT_STATES istate)
40  {
41      mlastState = mstate;
42      mstate = istate;
43  }
44
45  void DrivetrainStateMachine::doAutonMotion(std::function<void()> iaction) // disable
46  ↪ manual control, and execute the action
47  {
48      DT_STATES oldState = mstate;
49      setState(DT_STATES::busy);
50      iaction();
51      setState(oldState);
52  }
```



```

48 void DrivetrainStateMachine::controlState() // update the state based on controller
    ↪ input
49 {
50     if (mtoggleFieldCentric.changedToPressed()) // toggle field centric
51     {
52         if (mstate == DT_STATES::manual)
53         {
54             mstate = DT_STATES::fieldCentric;
55         }
56         else
57         {
58             mstate = DT_STATES::manual;
59         }
60     }
61 }
62
63 void DrivetrainStateMachine::update() // move the robot based on the state
64 {
65     switch (mstate)
66     {
67         case DT_STATES::off:
68             break;
69         case DT_STATES::busy:
70             break;
71         case DT_STATES::manual: // normal, arcade control
72             mdrivetrain.moveArcade(mcontroller.getAnalog(ControllerAnalog::leftY),
    ↪ mcontroller.getAnalog(ControllerAnalog::leftX),
    ↪ mcontroller.getAnalog(ControllerAnalog::rightX), false);
73             break;
74         case DT_STATES::fieldCentric: // field centric arcade control
75             QAngle direction = 90_deg -
    ↪ atan2(mcontroller.getAnalog(ControllerAnalog::leftY),
    ↪ mcontroller.getAnalog(ControllerAnalog::leftX)) * radian;
76             double magnitude = hypot(mcontroller.getAnalog(ControllerAnalog::leftX),
    ↪ mcontroller.getAnalog(ControllerAnalog::leftY));
77             mdrivetrain.moveInDirection(direction, true, magnitude,
    ↪ mcontroller.getAnalog(ControllerAnalog::rightX), true);
78             break;
79     }
80 }

```

3.12 src/util/CustomOdometry.cpp

```
1  /**
2   * CustomOdometry.cpp
3   *
4   * This file contains the declaration of the CustomOdometry class.
5   * CustomOdometry is responsible for doing all the math and storing
6   * information about the robot's position and orientation. Everything
7   * is static, because there doesn't need to be more than one position
8   * calculation.
9   */
10 #include "main.h" // gives access to CustomOdometry and other dependencies
11
12 /* ----- Private Information ----- */
13 /*                                     */
14 /* ----- */
15
16 /* ----- Constants ----- */
17 const double & CustomOdometry::moffFIn =
18     def::TRACK_FORWARD_OFFSET.convert(inch); // offset of forward tracking wheel in
19     ↪ inches
20 const double & CustomOdometry::moffSIn =
21     def::TRACK_SIDE_OFFSET.convert(inch); // offset of side tracking wheel in inches
22 const double & CustomOdometry::mcircIn =
23     def::TRACK_WHEEL_CIRCUMFERENCE.convert(inch); // tracking wheel circumference in
24     ↪ inches
25
26 /* ----- Sensor References ----- */
27 ADIEncoder & CustomOdometry::meF = def::track_encoder_forward; // left tracking wheel
28     ↪ encoder
29 ADIEncoder & CustomOdometry::meS = def::track_encoder_side; // right tracking wheel
30     ↪ encoder
31 pros::Imu & CustomOdometry::mimu1 = def::imu_bottom; // inertial sensors
32 pros::Imu & CustomOdometry::mimu2 = def::imu_top;
33
34 /* ----- Starting Values ----- */
35 OdomState CustomOdometry::mstate = {0_in, 0_in, 0_rad}; // position of the robot
36 bool CustomOdometry::menabled = true; // whether or not the loop is allowed to run
37
38 /* ----- Methods ----- */
39 std::valarray<double> CustomOdometry::getSensorVals() // returns new sensor values
40 {
41     return {meF.get(), meS.get(),
42             ((isinf(mimu1.get_rotation()) ? 0 : mimu1.get_rotation()) +
43              (isinf(mimu2.get_rotation()) ? 0 : mimu2.get_rotation())) *
44              M_PI / 180 / 2};
45 }
46
47 /* ----- Public Information ----- */
48 /*                                     */
49 /* ----- */
50 OdomState CustomOdometry::getState() { return mstate; } // returns the current state of
51     ↪ the robot
52 QLength CustomOdometry::getX() { return mstate.x; } // returns the x value of the state
```

```

48 | QLength CustomOdometry::getY() { return mstate.y; } // returns the y value of the state
49 | QAngle CustomOdometry::getTheta() { return mstate.theta; } // returns the theta value
    | ↪ of the state
50 | void CustomOdometry::setState(const OdomState & istate) // sets the state of the robot
51 | {
52 |     mstate = istate;
53 | }
54 |
55 | void CustomOdometry::enable() // allows the odometry thread to be started (but does not
    | ↪ start it)
56 | {
57 |     menabled = true;
58 | }
59 | void CustomOdometry::disable() // stops the odometry thread from running, prevents it
    | ↪ from starting
60 | {
61 |     menabled = false;
62 | }
63 |
64 | OdomState
65 | CustomOdometry::mathStep(const std::valarray<double>
66 |                          ivalDiff) // does one iteration of odometry math, given
    | ↪ sensor changes
67 | {
68 |     const double df =
69 |         ivalDiff[0] * mcircIn / 360; // stores the change of all tracking wheels in
    | ↪ inches
70 |     const double ds = ivalDiff[1] * mcircIn / 360;
71 |
72 |     double vectorLx, vectorLy; // declares local offset x and y
73 |     if (ivalDiff[2]) // if the robot turned
74 |     {
75 |         vectorLx = 2 * sin(ivalDiff[2] / 2) *
76 |             (ds / ivalDiff[2] + moffSIn); // sideways translation based on arc
77 |         vectorLy = 2 * sin(ivalDiff[2] / 2) *
78 |             (df / ivalDiff[2] + moffFIn); // forward translation based on arc
79 |     }
80 |     else
81 |     {
82 |         vectorLx = ds; // sideways translation (without turning)
83 |         vectorLy = df; // forward translation (without turning)
84 |     }
85 |
86 |     if (isnan(vectorLy)) // makes sure the local offsets exist
87 |         vectorLy = 0;
88 |     if (isnan(vectorLx))
89 |         vectorLx = 0;
90 |
91 |     double avgT =
92 |         mstate.theta.convert(radian) + ivalDiff[2] / 2; // calculates the direction
    | ↪ the robot moved
93 |
94 |     double polarR = hypot(vectorLx, vectorLy); // calculates polar coordinate, r
95 |     double polarT =

```

```

96         atan2(vectorLy, vectorLx) - avgT; // calculates polar coordinate, theta, and
           ↳ rotates
97
98     double dx = sin(polarT) * polarR; // converts new polar coordinates back to
           ↳ cartesian
99     double dy = cos(polarT) * polarR;
100
101     if (isnan(dx)) // makes sure the cartesian coordinates exist
102         dx = 0;
103     if (isnan(dy))
104         dy = 0;
105
106     return {dx * inch, dy * inch}; // return the change in position
107 }
108
109 /* ----- */
110 /*                      Friend Method                      */
111 /* ----- */
112 void odomTaskFunc(void *) // friend function to CustomOdometry to be run as a separate
           ↳ thread
113 {
114     std::valarray<double> lastVals{0, 0, 0},
115         newVals{0, 0, 0}; // arrays used for storing sensor values
116     OdomState newState; // used to store the change in state
117     waitForImu(); // wait for the inertial sensors to calibrate
118     CustomOdometry::mstate = def::SET_ODOM_START; // set the starting position to the
           ↳ origin
119
120     while (CustomOdometry::menabled)
121     {
122         newVals = CustomOdometry::getSensorVals(); // provides new sensor values and
           ↳ saves them
123         newState = CustomOdometry::mathStep(
124             newVals -
125             lastVals); // runs odometry math on sensor value change to calculate change
           ↳ in state
126         lastVals = newVals; // stores sensor values for the next iteration
127
128         // updates state based on change
129         CustomOdometry::mstate.x += newState.x;
130         CustomOdometry::mstate.y += newState.y;
131         CustomOdometry::mstate.theta = newVals[2] * radian;
132
133         pros::delay(10); // run odometry at 100hz (every 10 ms)
134     }
135 }

```

3.13 src/util/util.cpp

```
1  /**
2   * util.cpp
3   *
4   * This file contains miscellaneous utility functions and classes
5   * to help with the general organization of the rest of the code.
6   */
7  #include "main.h" // gives access to util.hpp and other dependencies
8
9  /* ----- */
10 /*                               ExtendedPoint Struct                               */
11 /* ----- */
12 * ExtendedPoint struct inherits from the built in okapi Point struct,
13 * but provides additional functionality, like an orientation value
14 * (theta) as well as x and y values. It also adds some vector operations that are used
15 ↪ for path
16 * following in the drivetrain class.
17 */
18 ExtendedPoint::ExtendedPoint(QLength ix, QLength iy, QAngle itheta) : theta(itheta)
19 {
20     x = ix;
21     y = iy;
22 }
23
24 /* ----- Subtraction ----- */
25 ExtendedPoint ExtendedPoint::operator-(const ExtendedPoint & ivec) // overloaded
26 ↪ subtraction
27 {
28     return ExtendedPoint(x - ivec.x, y - ivec.y, theta - ivec.theta);
29 }
30 ExtendedPoint ExtendedPoint::sub(const ExtendedPoint & ivec) // subtraction method
31 {
32     return *this - ivec;
33 }
34
35 /* ----- Addition ----- */
36 ExtendedPoint ExtendedPoint::operator+(const ExtendedPoint & ivec) // overloaded
37 ↪ addition
38 {
39     return ExtendedPoint(x + ivec.x, y + ivec.y, theta + ivec.theta);
40 }
41 ExtendedPoint ExtendedPoint::add(const ExtendedPoint & ivec) // addition method
42 {
43     return *this + ivec;
44 }
45
46 /* ----- Multiplication ----- */
47 QLength ExtendedPoint::dot(const ExtendedPoint & ivec) // dot multiply vectors
48 {
49     return (x.convert(inch) * ivec.x.convert(inch) + y.convert(inch) *
50 ↪ ivec.y.convert(inch)) * inch;
51 }
```

```

48 ExtendedPoint ExtendedPoint::operator*(const double iscalar) // overloaded scalar
   ↳ multiplication
49 {
50     return ExtendedPoint(x * iscalar, y * iscalar, theta * iscalar);
51 }
52 ExtendedPoint ExtendedPoint::scalarMult(const double iscalar) // multiply the vectors
   ↳ by a scalar
53 {
54     return *this * iscalar;
55 }
56 ExtendedPoint ExtendedPoint::operator*(const ExtendedPoint & ivec) // elementwise
   ↳ multiplication
57 {
58     return ExtendedPoint((x.convert(inch) * ivec.x.convert(inch)) * inch,
59                           (y.convert(inch) * ivec.y.convert(inch)) * inch,
60                           (theta.convert(degree) * ivec.theta.convert(degree)) * degree);
61 }
62 ExtendedPoint ExtendedPoint::eachMult(const ExtendedPoint & ivec) // elementwise
   ↳ multiplication
63 {
64     return *this * ivec;
65 }
66
67 /* ----- Comparative ----- */
68 bool ExtendedPoint::operator==(const ExtendedPoint & ipoint) // overloaded equivalence
   ↳ check
69 {
70     return x == ipoint.x && y == ipoint.y && theta == ipoint.theta;
71 }
72
73 /* ----- Other ----- */
74 QLength ExtendedPoint::dist(const ExtendedPoint & ivec) // distance between points
75 {
76     return sqrt((x - ivec.x).convert(inch) * (x - ivec.x).convert(inch) +
77                 (y - ivec.y).convert(inch) * (y - ivec.y).convert(inch)) *
78             inch;
79 }
80 QLength ExtendedPoint::mag() // magnitude
81 {
82     return sqrt(x.convert(inch) * x.convert(inch) + y.convert(inch) * y.convert(inch))
83             ↳ * inch;
84 }
85 ExtendedPoint ExtendedPoint::normalize() // creates a vector with a length of 1
86 {
87     return ExtendedPoint((x.convert(inch) / mag().convert(inch)) * inch,
88                           (y.convert(inch) / mag().convert(inch)) * inch, theta);
89 }
90 std::string ExtendedPoint::string() // returns the point in string form for testing
91 {
92     return "{" + std::to_string(x.convert(inch)).substr(0, 3) + ", " +
93             std::to_string(y.convert(inch)).substr(0, 3) + ", " +
94             std::to_string(theta.convert(degree)).substr(0, 3) + "}";
95 }

```

```

96  /* ----- Misc Functions ----- */
97  /*                               */
98  /* ----- Misc Functions ----- */
99  void waitForImu() // blocks the execution of the code until the imu is done calibrating
100 {
101     while (def::imu_top.is_calibrating() || def::imu_bottom.is_calibrating())
102         pros::delay(100);
103 }
104
105 /* ----- OdomDebug Helpers ----- */
106 void odomSetState(OdomDebug::state_t istate) // sets the state of odometry based on
    ↳ display inputs
107 {
108     CustomOdometry::setState({istate.x, istate.y, istate.theta});
109 }
110 void odomResetAll() // resets everything having to do with odometry (for "Reset" button)
111 {
112     CustomOdometry::setState({0_ft, 0_ft, 0_deg}); // sets the robot's position to 0
113     def::imu_top.reset(); // resets the imu
114     def::imu_bottom.reset(); // resets the imu
115     // resets the encoders
116     def::track_encoder_forward.reset();
117     def::track_encoder_side.reset();
118     waitForImu(); // waits for the imu
119 }
120
121 /* ----- Task Functions ----- */
122 void sm_dt_task_func(void *) // state machine drivetrain task to be run independently
123 {
124     while (true)
125     {
126         def::sm_dt.controlState(); // update the state from controller input
127         def::sm_dt.update(); // moves the robot based on the state
128         pros::delay(20);
129     }
130 }
131
132 void sm_bc_task_func(void *) // state machine ball control task to be run independently
133 {
134     while (true)
135     {
136         if (def::sm_bc.controlEnabled)
137         {
138             def::sm_bc.controlState(); // update the state from controller input if it
                ↳ is enabled
139         }
140         def::sm_bc.update(); // moves the robot based on the state
141         pros::delay(20);
142     }
143 }
144
145 void display_task_func(void *) // display task to be run independently
146 {
147     while (true)

```

```

148     {
149         def::display.setOdomData(); // update the odometry information
150
151         // room for any other miscellaneous debugging
152
153         pros::delay(20);
154     }
155 }
156
157 /* ----- Macros ----- */
158 void deploy() // deploys the robot
159 {
160     def::sm_bc.fwShoot(); // deploys the hood
161     pros::delay(250);
162     def::sm_bc.fwOff();
163 }
164
165 /* ----- Control ----- */
166 /* ----- Control ----- */
167 /* ----- Control ----- */
168
169 /* ----- PID Class ----- /
170 * PID is a feedback loop that uses the difference between the goal and the current
171 ↪ position (error)
172 * of the robot to decide how much power to give the motors. The "P" stands for
173 ↪ "proportional", and
174 * it adds power proportional to the error, so it gets slower and slower as it gets
175 ↪ closer to the
176 * goal to prevent it from driving too fast past it. The "D" stands for "derivative",
177 ↪ because it
178 * uses the derivative of the error (the speed of the robot) to apply power. The faster
179 ↪ the robot
180 * goes, the more the d term works to slow it down. The "I" stands for "integral",
181 ↪ because it uses
182 * the integral of the error (the absement of the robot) to apply power. When the robot
183 ↪ is close to
184 * the goal, sometimes the "P" and "D" terms do not apply enough power to move the
185 ↪ robot, but when
186 * the robot isn't moving (and when it is), the "I" term is accumulating, so it
187 ↪ eventually builds up
188 * enough to move the robot even closer to the goal. This implementation of PID only
189 ↪ enables the "I"
190 * term when the robot is close enough to the goal, to prevent "integral windup", which
191 ↪ is when the
192 * integral gets too big when it's too far away from the goal.
193 *
194 * We have a PID controller class, because we use different PID loops in many different
195 ↪ places in
196 * the code, so we wanted to be able to be able to quickly make one with constants
197 ↪ specific to the
198 * application.
199 */
200 PID::PID(double ikP, double ikI, double ikD, double ikIRange, double isettlerError,
201          double isettlerDerivative,

```



```

189         QTime isettlerTime) // constructor that sets constants, and initializes
190         ↪ variables
191     : msettlerError(isettlerError), msettlerDerivative(isettlerDerivative),
192       msettlerTime(isettlerTime), msettler(TimeUtilFactory::withSettledUtilParams(
193           msettlerError, msettlerDerivative,
194           ↪ msettlerTime)
195           .getSettledUtil()),
196       mkP(ikP), mkI(ikI), mkD(ikD), mkIRange(ikIRange), merror(0), mlastError(0),
197       ↪ mtotalError(0),
198       mderivative(0)
199 {
200 }
201
202 PID::PID(const PID & ioother) // copy constructor for duplicating PID objects behind the
203 ↪ scenes
204 {
205     msettlerError = ioother.msettlerError;
206     msettlerDerivative = ioother.msettlerDerivative;
207     msettlerTime = ioother.msettlerTime;
208     msettler =
209         TimeUtilFactory::withSettledUtilParams(msettlerError, msettlerDerivative,
210             ↪ msettlerTime)
211         .getSettledUtil();
212     mkP = ioother.mkP;
213     mkI = ioother.mkI;
214     mkD = ioother.mkD;
215     mkIRange = ioother.mkIRange;
216     mlastError = ioother.mlastError;
217     mtotalError = ioother.mtotalError;
218     mderivative = ioother.mderivative;
219 }
220
221 double PID::getLastError() { return mlastError; }
222 double PID::getTotalError() { return mtotalError; }
223
224 void PID::setGains(double ikP, double ikI, double ikD) // used only for changing
225 ↪ constants later
226 {
227     mkP = ikP;
228     mkI = ikI;
229     mkD = ikD;
230 }
231
232 double PID::getP() { return mkP; }
233 double PID::getI() { return mkI; }
234 double PID::getD() { return mkD; }
235
236 double PID::iterate(double ierror) // goes through one iteration of the PID loop
237 {
238     merror = ierror;
239     if (mkI != 0) // regulate integral term
240     {
241         if (abs(merror) < mkIRange && merror != 0) // if in range, update mtotalError
242         {

```

```

237         mtotalError += merror;
238     }
239     else
240     {
241         mtotalError = 0;
242     }
243     util::chop<double>(-50 / mkI, 50 / mkI,
244                       mtotalError); // limit mtotalError to prevent integral windup
245 }
246
247 mderivative = merror - mlastError; // calculate the derivative before lastError is
    ↳ overwritten
248 mlastError = merror; // save the current error for the next cycle
249
250 return merror * mkP + mtotalError * mkI + mderivative * mkD;
251 }
252
253 bool PID::isSettled() // returns whether or not the controller is settled at the target
254 {
255     return msettler->isSettled(merror);
256 }
257
258 /* ----- Slew Class ----- /
259 * Slew rate control is a system that limits the change in speed to prevent wheel slip.
    ↳ If the robot
260 * changes speed too fast, the wheels can slip, and make the robot's motion less fluid.
    ↳ When the
261 * target speed changes by a lot, the slew rate controller slowly increases it's output
    ↳ to
262 * eventually get to the target speed.
263 *
264 * This Slew rate controller is also intended to be used with PID, but sometimes when
    ↳ slew is used
265 * with PID, it interferes with the settling of the PID. To prevent this, the slew rate
    ↳ controller
266 * is only active when there are large changes in the target input value, making it
    ↳ only really
267 * affect the beginning of the motion. For example, if the motors aren't moving, and
    ↳ the target
268 * value suddenly jumps to 100%, the slew controller might gradually increase by
    ↳ increments of 5%
269 * until it reaches 100%, but if the target value jumps to from 0% to 20%, the slew
    ↳ controller might
270 * not engage (actual values depend on constants "mincrement" and "mactiveDifference").
271 */
272 Slew::Slew(double iincrement, double iactiveDifference)
273     : mincrement(iincrement), mactiveDifference(iactiveDifference), mlastValue(0) //
    ↳ constructor
274 {
275 }
276
277 double Slew::getIncrement() { return mincrement; }
278 double Slew::getActiveDifference() { return mactiveDifference; }
279 double Slew::getLastValue() { return mlastValue; }

```

```

280
281 double Slew::iterate(double ivalue) // limits the input value to maximum changes
↳ described by
282                                     // constants when run in a loop
283 {
284     if (abs(ivalue - mlastValue) >
285         mactiveDifference) // only activate if the value difference is over the
↳ threshold
286     {
287         if (ivalue >
288             mlastValue +
289             mincrement) // if the input is too big, only let it increase by a
↳ maximum amount
290         {
291             mlastValue = mlastValue + mincrement;
292             return mlastValue;
293         }
294         else if (ivalue < mlastValue - mincrement) // if the input is too small, only
↳ let it
295                                     // decrease by a maximum amount
296         {
297             mlastValue = mlastValue - mincrement;
298             return mlastValue;
299         }
300     }
301     mlastValue = ivalue;
302     return ivalue; // this only happens if nothing is wrong
303 }
304
305 /* ----- */
306 /*                               Util                               */
307 /* ----- /
308 * The util namespace is used to organaize basic functions that don't
309 * necessarily need to be used for robotics.
310 */
311
312 /* ----- Angle Wrappers ----- /
313 * These methods take any angle, and return an angle representing the same position in
↳ a specific
314 * range. For example, wrapDeg(370) returns 10, because 370 is out of the range [0,
↳ 360).
315 */
316 double util::wrapDeg(double iangle) // range [0, 360)
317 {
318     iangle = fmod(iangle, 360);
319     if (iangle < 0)
320         iangle += 360;
321     return iangle;
322 }
323 double util::wrapDeg180(double iangle) // range [-180, 180]
324 {
325     iangle = fmod(iangle, 360);
326     if (iangle < -180)
327         iangle += 360;

```

```

328     else if (iangle > 180)
329         iangle -= 360;
330     return iangle;
331 }
332 double util::wrapRad(double iangle) // range [0, 2pi)
333 {
334     iangle = fmod(iangle, 2 * 3.14159265358979323846);
335     if (iangle < 0)
336         iangle += 2 * 3.14159265358979323846;
337     return iangle;
338 }
339 double util::wrapRadPI(double iangle) // range [-pi, pi]
340 {
341     iangle = fmod(iangle, 2 * 3.14159265358979323846);
342     if (iangle < -3.14159265358979323846)
343         iangle += 2 * 3.14159265358979323846;
344     else if (iangle > 3.14159265358979323846)
345         iangle -= 2 * 3.14159265358979323846;
346     return iangle;
347 }
348 QAngle util::wrapQAngle(QAngle iangle) // range [0, 360) for QAngles
349 {
350     iangle = fmod(iangle.convert(degree), 360) * degree;
351     if (iangle < 0_deg)
352         iangle += 360_deg;
353     return iangle;
354 }
355 QAngle util::wrapQAngle180(QAngle iangle) // range [-180, 180] for QAngles
356 {
357     iangle = fmod(iangle.convert(degree), 360) * degree;
358     if (iangle < -180_deg)
359         iangle += 360_deg;
360     else if (iangle > 180_deg)
361         iangle -= 180_deg;
362     return iangle;
363 }

```