

```

1 #include "includes.h"
2 #include <array>
3
4
5 //subsystem objects
6 Drive *drive = new Drive();
7 Pneumatics *fourbarpneum = new Pneumatics(FRONT_PNEUM);
8 Pneumatics *backclamppneum = new Pneumatics(BACK_PNEUM);
9 Effectors effectors;
10 Intake *intake = new Intake(INTAKE_PORT);
11 Intake *fourbar1 = new Intake(FOUR_BAR_FIRST);
12 Intake *fourbar2 = new Intake(FOUR_BAR_FIRST);
13 Button *buttons = new Button();
14 pros::Imu imu(IMU_PORT);
15 pros::ADIDigitalIn but('H');
16 okapi::Controller controller (okapi::ControllerId::master);
17
18 double speeds[3] = {150, 150, 150};
19
20 PIDConst forwardDefault = {0.035, 0.00005, 0};
21 PIDConst headingDefault = {0, 0, 0};
22 PIDConst turnDefault = {0.02, 0.000025, 0};
23
24 int route;
25
26 /**
27  * A callback function for LLEMU's center button.
28  *
29  * When this callback is fired, it will toggle line 2 of the LCD text between
30  * "I was pressed!" and nothing.
31  */
32 void on_center_button() {
33     static bool pressed = false;
34     pressed = !pressed;
35     if (pressed) {
36         pros::lcd::set_text(2, "I was pressed!");
37     } else {
38         pros::lcd::clear_line(2);
39     }
40 }
41
42 void autonSelector() {
43     while(1) {
44         if(controller.getDigital(okapi::ControllerDigital::A)) {
45             route = 1;
46             break;
47         }
48         if(controller.getDigital(okapi::ControllerDigital::B)) {
49             route = 2;
50             break;
51         }
52         if(controller.getDigital(okapi::ControllerDigital::X)) {
53             route = 3;
54             break;
55         }
56         if(controller.getDigital(okapi::ControllerDigital::Y)) {
57             route = 4;

```

```

58         break;
59     }
60     pros::delay(15);
61 }
62 }
63
64
65 /**
66  * Runs initialization code. This occurs as soon as the program is started.
67  *
68  * All other competition modes are blocked by initialize; it is recommended
69  * to keep execution time for this mode under a few seconds.
70  */
71
72
73 void initialize() {
74     //make sure four bar can't go higher/lower than the mechanical stops
75     fourbar1->setLimits(2400, 0);
76     fourbar2->setLimits(2400, 0);
77
78     pros::lcd::initialize();
79     pros::lcd::set_text(1, "Hello PROS User!");
80
81     //autonSelector();
82
83     //calibrate imu
84     imu.reset();
85
86     int time = pros::millis();
87     int iter = 0;
88     while (imu.is_calibrating())
89     {
90         printf("IMU calibrating... %d\n", iter);
91         iter += 10;
92         pros::delay(10);
93     }
94     imu.set_data_rate(10);
95     pros::lcd::register_btn1_cb(on_center_button);
96 }
97
98 //x-drive math to find difference between two odomstates
99 OdomState transform(OdomState curr, OdomState target) {
100     OdomState diff;
101     diff.x = target.x - curr.x;
102     diff.y = target.y - curr.y;
103     diff.theta = OdomMath::constrainAngle180(target.theta-curr.theta);
104
105
106     //rotate vector
107     diff.x = diff.y * sin(curr.theta.convert(radian)) +
diff.x*cos(curr.theta.convert(radian));
108     diff.y = diff.y*cos(curr.theta.convert(radian)) - diff.x*sin(curr.theta.convert(radian));
109
110     return diff;
111 }
112
113

```

```

114 //slew limiter for PID
115 double limiter(double prevOutput, double currOutput, double step) {
116     double output;
117     if(currOutput > 0){ // positive rawOutput case
118
119         output = std::clamp(currOutput, prevOutput - step, std::min(1.0, prevOutput + step));
120         // clamped for slew and so finalOutput does not exceed maxOutput
121         output = std::max(0.2, output); //make sure output above min power
122     } else if (currOutput < 0){ // negative rawOutput case
123
124         output = std::clamp(currOutput, std::max(-1.0, prevOutput - step), prevOutput +
125 step); // clamped for slew and so finalOutput does not exceed -maxOutput
126         output = std::min(-0.2, output); //make sure output above min power
127     } else { // rawOutput is 0
128
129         output = 0; // step will return 0
130     }
131     return output;
132 }
133
134
135 //overhauled move function
136 void pidMoveForward(OdomState target, PIDConst forwardConstants, PIDConst headingConstants,
137 double timeout = 5) {
138     double forward, turn, prevForward, prevTurn;
139     QLength magerr;
140     QAngle headerr;
141     QAngle targetAngle;
142     OdomState currState;
143     QLength xDiff, yDiff;
144     prevForward = 0;
145     prevTurn = 0;
146
147     PID forwardObj = PID(forwardConstants);
148     PID turnObj = PID(headingConstants);
149
150     double startTime = pros::millis();
151     do {
152         currState = drive->getState();
153         xDiff = target.x-currState.x;
154         yDiff = target.y-currState.y;
155
156         targetAngle = okapi::OdomMath::constrainAngle180((PI/2 - atan2(xDiff.convert(meter),
157 yDiff.convert(meter)))*1_rad);
158         targetAngle = 1_deg * targetAngle.convert(degree);
159
160         //calculate errors
161         QAngle curr = okapi::OdomMath::constrainAngle180(imu.get_heading()*1_deg);
162         headerr = okapi::OdomMath::constrainAngle180(curr-targetAngle);
163         magerr = sqrt((xDiff * xDiff) + (yDiff * yDiff));
164
165         if(abs(magerr.convert(inch))<10) {
166             headerr = 0_deg;
167         }
168
169         //if overshoot point, reverse direction and target heading

```

```

168     if(abs(headerr.convert(degree)) > 100) {
169         headerr = okapi::OdomMath::constrainAngle180(headerr-180_deg);
170         magerr*=-1;
171     }
172
173     //limit and set motors
174     forward = limiter(prevForward, forwardObj.step(magerr.convert(inch)), 0.11);
175     turn = limiter(prevTurn, turnObj.step(headerr.convert(degree)), 0.11);
176     printf("%f %f %f\n", drive->getX(), drive->getY(), drive->getHeading());
177     drive->runTankArcade(forward, turn);
178     prevForward = forward;
179     prevTurn = turn;
180     pros::delay(10);
181 } while((abs(magerr.convert(inch)) > 2) && pros::millis()-startTime > (timeout*1000) );
//timeout and check mag err
182     drive->runTankArcade(0, 0);
183 }
184
185 //overhauled turn function
186 void pidTurn(QAngle targetHeading, PIDConst turnConstants, double timeout = 3) {
187     double turn, prevTurn;
188     QAngle headerr;
189     QAngle targetAngle;
190     OdomState currState;
191     QLength xDiff, yDiff;
192     prevTurn = 0;
193
194     targetAngle = targetHeading;
195
196     PID turnObj = PID(turnConstants);
197     do {
198         currState = drive->getState();
199
200
201         //calculate errors
202         QAngle curr = okapi::OdomMath::constrainAngle180(imu.get_heading()*1_deg);
203         headerr = okapi::OdomMath::constrainAngle180(curr-targetAngle);
204
205
206         //limit and set motors
207         turn = limiter(prevTurn, turnObj.step(headerr.convert(degree)), 0.11);
208         printf("%f %f %f %f\n", drive->getX(), drive->getY(), drive->getHeading(),
abs(headerr.convert(degree))>3);
209         drive->runTankArcade(0, turn);
210         prevTurn = turn;
211         pros::delay(10);
212     } while(abs(headerr.convert(degree))>3);
213     drive->runTankArcade(0, 0);
214 }
215
216 //move to any point
217 void moveToPoint(OdomState target, PIDConst forwardConstants, PIDConst headingConstants,
PIDConst turnConstants, double timeoutforward = 5, double timeoutturn = 3) {
218     OdomState currState = drive->getState();
219     QLength xDiff = target.x-currState.x;
220     QLength yDiff = target.y-currState.y;
221
222     QAngle targetAngle = okapi::OdomMath::constrainAngle180((PI/2 -

```

```

    atan2(xDiff.convert(meter), yDiff.convert(meter))*1_rad);
223     targetAngle = 1_deg * targetAngle.convert(degree);
224
225     pidTurn(targetAngle, turnConstants, timeoutturn);
226     printf("Done turning");
227     pidMoveForward(target, forwardConstants, headingConstants, timeoutforward);
228 }
229
230 //PID move function that can handle turns and forward movements
231 void pidMoveTank(OdomState target, PIDConst forwardConstants = forwardDefault, PIDConst
turnConstants = headingDefault, bool turning = false) {
232     double forward, turn, prevForward, prevTurn;
233     QLength magerr;
234     QAngle headerr;
235     QAngle targetAngle;
236     OdomState currState;
237     QLength xDiff, yDiff;
238     prevForward = 0;
239     prevTurn = 0;
240
241     //forward and turn objects for PID
242     PID forwardObj = PID(forwardConstants);
243     PID turnObj = PID(turnConstants);
244
245     do {
246         currState = drive->getState();
247         xDiff = target.x-currState.x;
248         yDiff = target.y-currState.y;
249
250         //calculate target thetas differently depending on forward or turn
251         if(!turning) {
252             targetAngle = okapi::OdomMath::constrainAngle180((PI/2 -
atan2(xDiff.convert(meter), yDiff.convert(meter))*1_rad);
253             targetAngle = 1_deg * targetAngle.convert(degree);
254         }
255         else {
256             targetAngle = target.theta;
257         }
258
259         //calculate errors
260         QAngle curr = okapi::OdomMath::constrainAngle180(imu.get_heading()*1_deg);
261         headerr = okapi::OdomMath::constrainAngle180(curr-targetAngle);
262         magerr = sqrt((xDiff * xDiff) + (yDiff * yDiff));
263
264         //if overshoot point, reverse direction and target heading
265         if(abs(headerr.convert(degree)) > 100 && forwardConstants.kp != 0) {
266             headerr = okapi::OdomMath::constrainAngle180(headerr-180_deg);
267             magerr*=-1;
268         }
269
270         //limit and set motors
271         forward = limiter(prevForward, forwardObj.step(magerr.convert(inch)), 0.11);
272         turn = limiter(prevTurn, turnObj.step(headerr.convert(degree)), 0.11);
273         printf("%f %f %f\n", drive->getX(), drive->getY(), drive->getHeading());
274         drive->runTankArcade(forward, turn);
275         prevForward = forward;
276         prevTurn = turn;
277         pros::delay(10);

```

```

278     } while((abs(magerr.convert(inch)) > 3 && !turning) || (abs(headerr.convert(degree))>3 &&
turning)); //tolerances checked differently depending on turning or forward
279     drive->runTankArcade(0, 0);
280 }
281
282
283 //use odometry magnitude error to move a set distance
284 void distanceMove(double distance, double speed) {
285     OdomState initial = drive->getState();
286     double error = 0;
287     // double start = drive->getEncoder();
288     drive->runTankArcade(speed, 0);
289     double start = pros::millis();
290     do {
291         OdomState temp = drive->getState();
292         QLength xdiff = temp.x-initial.x;
293         QLength ydiff = temp.y-initial.y;
294         printf("Odom: %f %f %f\n", temp.x.convert(inch), temp.y.convert(inch),
temp.theta.convert(degree));
295         // error = ((drive->getEncoder() - start) / 360*(7.0/5)) * 4 * PI ;
296         error = okapi::sqrt((xdiff*xdiff) + (ydiff*ydiff)).convert(inch);
297         pros::delay(30);
298     } while(error<distance && pros::millis()-start<6000);
299     drive->runTankArcade(0, 0);
300 }
301
302 void distancePID(double distance, PIDConst gains) {
303     OdomState initial = drive->getState();
304     double error = 0;
305     double speed = 0;
306     double prevSpeed = 0;
307     PID obj = PID(gains);
308     do {
309         OdomState temp = drive->getState();
310         QLength xdiff = temp.x-initial.x;
311         QLength ydiff = temp.y-initial.y;
312         printf("Odom: %f %f %f\n", temp.x.convert(inch), temp.y.convert(inch),
temp.theta.convert(degree));
313         error = okapi::sqrt((xdiff*xdiff) + (ydiff*ydiff)).convert(inch);
314         speed = limiter(prevSpeed, obj.step(distance>0 ? distance-error: distance+error),
0.11);
315         prevSpeed = speed;
316         drive->runTankArcade(speed, 0);
317         pros::delay(30);
318     } while(error<abs(distance));
319     drive->runTankArcade(0, 0);
320 }
321
322 void speedMove(double time, double speed) {
323
324     double start = pros::millis();
325     printf("%f\n", start);
326     drive->runTankArcade(speed, 0);
327     while(pros::millis()-start<time) {
328         pros::delay(10);
329     }
330     double end = pros::millis();
331     printf("%f\n", end);

```

```

332     drive->runTankArcade(0, 0);
333 }
334
335 void setEffectorPositions() {
336     //set all effector positions
337     effectors.addPosition();
338 }
339
340 void turnTest() {
341     OdomState goal = drive->getState();
342     while(1) {
343         goal.theta = 90_deg;
344         pidMoveTank(goal, {0, 0, 0}, {0.007, 0.000005, 0}, true);
345         pros::delay(1000);
346         goal = drive->getState();
347         goal.theta = 180_deg;
348         pidMoveTank(goal, {0, 0, 0}, {0.007, 0.000005, 0}, true);
349         pros::delay(1000);
350         goal = drive->getState();
351         goal.theta = 270_deg;
352         pidMoveTank(goal, {0, 0, 0}, {0.007, 0.000005, 0}, true);
353     }
354 }
355
356
357 //drag turn function
358 void dragTurn(double heading, double direction, double side) {
359     while(abs(heading-imu.get_heading())>2) {
360         //handle side and direction logic
361         if(side == 0) {
362             drive->runTank(0.5*direction, 0.1*direction*-1);
363         }
364         if(side == 1) {
365             drive->runTank(0.1*direction*-1, 0.5*direction);
366         }
367         pros::delay(10);
368     }
369     drive->runTank(0, 0);
370 }
371
372
373 //experimental autonSelector function
374
375
376 void autobalancer(double tolerance) {
377     drive->runTankArcade(-1, 0);
378     pros::delay(2000);
379     double curr_pitch = imu.get_pitch();
380     double last_pitch = curr_pitch;
381     while(curr_pitch > tolerance || curr_pitch-last_pitch>=0) {
382         pros::delay(5);
383         printf("Current pitch: %f\n", curr_pitch);
384         last_pitch = curr_pitch;
385         curr_pitch = imu.get_pitch();
386     }
387     drive->runTankArcade(0, 0);
388     drive->setMode(okapi::AbstractMotor::brakeMode::hold);

```

```

389     pros::delay(10000);
390 }
391
392 void testBalancing() {
393     setEffectorPositions();
394
395     fourbarpneum->turnOn();
396     fourbar1->moveTarget(2400);
397     fourbar2->moveTarget(2400);
398     pros::delay(2000);
399     fourbar1->moveTarget(0);
400     fourbar2->moveTarget(0);
401
402     pros::delay(2000);
403     autobalancer(22);
404 }
405
406 void moveUntilButton(double speed) {
407     drive->runTankArcade(speed, 0);
408     while(!but.get_value()) {
409         pros::delay(10);
410     }
411     drive->runTankArcade(0, 0);
412 }
413 /**
414  * Runs while the robot is in the disabled state of Field Management System or
415  * the VEX Competition Switch, following either autonomous or opcontrol. When
416  * the robot is enabled, this task will exit.
417  */
418 void disabled() {}
419
420 /**
421  * Runs after initialize(), and before autonomous when connected to the Field
422  * Management System or the VEX Competition Switch. This is intended for
423  * competition-specific initialization routines, such as an autonomous selector
424  * on the LCD.
425  *
426  * This task will exit when the robot is enabled and autonomous or opcontrol
427  * starts.
428  */
429 void competition_initialize() {}
430
431 /**
432  * Runs the user autonomous code. This function will be started in its own task
433  * with the default priority and stack size whenever the robot is enabled via
434  * the Field Management System or the VEX Competition Switch in the autonomous
435  * mode. Alternatively, this function may be called in initialize or opcontrol
436  * for non-competition testing purposes.
437  *
438  * If the robot is disabled or communications is lost, the autonomous task
439  * will be stopped. Re-enabling the robot will restart the task, not re-start it
440  * from where it left off.
441  */
442
443 /**
444  * Runs the operator control code. This function will be started in its own task
445  * with the default priority and stack size whenever the robot is enabled via

```



```

446 * the Field Management System or the VEX Competition Switch in the operator
447 * control mode.
448 *
449 * If no competition control is connected, this function will run immediately
450 * following initialize().
451 *
452 * If the robot is disabled or communications is lost, the
453 * operator control task will be stopped. Re-enabling the robot will restart the
454 * task, not resume it from where it left off.
455 */
456
457 void driverMovementTrack() {
458     //initialize variables and set effector positions
459     setEffectorPositions();
460     int parking = 0;
461     double forward;
462     double turn;
463     double strafe;
464     int i = 0;
465     bool fourbarpneumstate = true;
466     bool backclampstate = false;
467     ADIEncoder righttrack = ADIEncoder('A', 'B', false); //encoders because i don't
know how to get values
468     ADIEncoder lefttrack = ADIEncoder('C', 'D', true);
469     righttrack.reset();
470     lefttrack.reset();
471     okapi::Motor fourbar(FOUR_BAR_FIRST);
472
473     double max = 1;
474     drive->setMode(okapi::AbstractMotor::brakeMode::hold);
475     fourbarpneum->turnOn();
476     backclamppneum->turnOff();
477     while(true) {
478         //toggle between coast and hold brake modes
479         //get controller and drive chassis base
480         // printf("%f %f\n", righttrack.get(), lefttrack.get());
481         // printf("%f %f %d\n", drive->getX(), drive->getY(), (int)drive->getHeading()%360);
482         //update all button values
483         buttons->handleButtons(controller);
484         int buttonCounts[9];
485         for(int i = 0; i < 9; i++) {
486             buttonCounts[i] = buttons->getCount(buttons->buttonList[i]);
487         }
488
489         if(buttonCounts[7]%2) {
490             max = 5.0/7;
491             drive->setMode(okapi::AbstractMotor::brakeMode::hold);
492         }
493         else {
494             max = 1;
495             drive->setMode(okapi::AbstractMotor::brakeMode::coast);
496         }
497
498         forward = controller.getAnalog(okapi::ControllerAnalog::leftY);
499         turn = controller.getAnalog(okapi::ControllerAnalog::rightX);
500         if(forward>=0) {
501             drive->runTankArcade(std::max(forward*-(6.0/7), max*-1), turn*-0.6);

```

```

502     }
503     else {
504         drive->runTankArcade(std::min(forward*-(6.0/7), max), turn*-0.6);
505     }
506
507     // effectors.step(buttonCounts, speeds); //handle two bar
508
509     //intake->run(false, buttons->getPressed(okapi::ControllerDigital::right), 150);
//handle intake
510     //runs the intake backwards
511     if (buttonCounts[8]%2 == 1) {
512         intake->handle(buttonCounts[8], -180);
513     }
514     else {
515         intake->handle(buttonCounts[3], 180); //handle intake (toggle)
516     }
517
518     //handle four bar
519     fourbar1->run(buttons->getPressed(okapi::ControllerDigital::R1), buttons-
>getPressed(okapi::ControllerDigital::R2), 175);
520     //fourbar2->run(buttons->getPressed(okapi::ControllerDigital::R1), buttons-
>getPressed(okapi::ControllerDigital::R2), 175);
521
522     //handle clamp
523     fourbarpneum->handle(buttonCounts[5]);
524     backclamppneum->handle(buttonCounts[0]);
525     // parking = buttonCounts[7] % 2;
526     // if (parking == 1) {
527     //     drive->setMode(okapi::AbstractMotor::brakeMode::hold);
528     // } else {
529     //     drive->setMode(okapi::AbstractMotor::brakeMode::coast);
530     // }
531     //printf("%d\n", parking);
532
533     if (buttonCounts[1]%2) {
534         printf("Heading: %f    Distance from last point: %f inches    Four Bar position:
%f\n\n", imu.get_heading(), ((righttrack.get() + lefttrack.get())/2 * 2.81665 * PI / 360),
fourbar.getEncoder());
535         righttrack.reset();
536         lefttrack.reset();
537     }
538
539     pros::delay(60);
540     pros::lcd::clear_line(2);
541 }
542 }
543
544 void opcontrol() {
545
546
547
548     //initialize variables and set effector positions
549     setEffectorPositions();
550     int parking = 0;
551     double forward;
552     double turn;
553     double strafe;
554     int i = 0;

```

```

555     bool fourbarpneumstate = true;
556     bool auxilclampstate = false;
557     ADIEncoder righttrack = ADIEncoder('A', 'B', false);           //encoders because i don't
know how to get values
558     ADIEncoder lefttrack = ADIEncoder('C', 'D', true);
559
560     double max = 1;
561     drive->setMode(okapi::AbstractMotor::brakeMode::hold);
562     while(true) {
563         //toggle between coast and hold brake modes
564         //get controller and drive chassis base
565         printf("%f %f\n", righttrack.get(), lefttrack.get());
566         // printf("%f %f %d\n", drive->getX(), drive->getY(), (int)drive->getHeading()%360);
567         //update all button values
568         buttons->handleButtons(controller);
569         int buttonCounts[9];
570         for(int i = 0; i < 9; i++) {
571             buttonCounts[i] = buttons->getCount(buttons->buttonList[i]);
572         }
573
574         if(buttonCounts[7]%2) {
575             max = 5.0/7;
576         }
577         else {
578             max = 1;
579         }
580
581         forward = controller.getAnalog(okapi::ControllerAnalog::leftY);
582         turn = controller.getAnalog(okapi::ControllerAnalog::rightX);
583         if(forward>=0) {
584             drive->runTankArcade(std::max(forward*-(6.0/7), max*-1), turn*-.0.6);
585         }
586         else {
587             drive->runTankArcade(std::min(forward*-(6.0/7), max), turn*-.0.6);
588         }
589
590         // effectors.step(buttonCounts, speeds); //handle two bar
591
592         //intake->run(false, buttons->getPressed(okapi::ControllerDigital::right), 150);
//handle intake
593         if (buttonCounts[8]%2 == 1) {
594             intake->handle(buttonCounts[8], -200);
595         }
596         else {
597             intake->handle(buttonCounts[3], 200); //handle intake (toggle)
598         }
599
600         //handle four bar
601         fourbar1->run(buttons->getPressed(okapi::ControllerDigital::R1), buttons-
>getPressed(okapi::ControllerDigital::R2), 175);
602         //fourbar2->run(buttons->getPressed(okapi::ControllerDigital::R1), buttons-
>getPressed(okapi::ControllerDigital::R2), 175);
603
604         //handle clamp
605         fourbarpneum->handle(buttonCounts[5]);
606         backclamppneum->handle(buttonCounts[0]);
607         parking = buttonCounts[7] % 2;
608         if (parking == 1){

```

```

609         drive->setMode(okapi::AbstractMotor::brakeMode::hold);
610     } else {
611         drive->setMode(okapi::AbstractMotor::brakeMode::coast);
612     }
613     //printf("%d\n", parking);
614     pros::delay(60);
615     pros::lcd::clear_line(2);
616 }
617 }
618
619 void right() {
620     setEffectorPositions();
621     printf("done\n");
622     distanceMove(39, -1); //move towards side neutral at full speed
623     fourbarpneum->turnOn(); //clamp it
624     pros::delay(100);
625     printf("Finished\n");
626     distanceMove(10, 1); //move back
627     distanceMove(4, 0.6);
628
629
630     pidTurn(270_deg, {0.01, 0.000008, 0}); //make the turn
631
632     drive->runTankArcade(0.5, 0); //move towards alliance goal
633     pros::delay(1400);
634     backclamppneum->turnOn();
635     drive->runTankArcade(-0.5, 0);
636     pros::delay(1000);
637     drive->runTankArcade(0, 0);
638     intake->run(true, false, -180);
639
640
641 }
642 }
643
644 void leftfast() {
645     setEffectorPositions();
646
647     distanceMove(42, -1); //move towards side neutral at full speed
648     fourbarpneum->turnOn(); //clamp it
649     printf("Finished\n");
650     distanceMove(30, 1); //move back
651 }
652
653 void thenewnewskills() {
654
655     //first alliance pickup
656     setEffectorPositions();
657     distanceMove(10, -0.5);
658     pidTurn(270_deg, {0.007, 0.000008, 0});
659     drive->runTankArcade(0.5, 0);
660     pros::delay(1000);
661     backclamppneum->turnOn();
662     pidTurn(270_deg, {0.007, 0.000008, 0});
663     drive->runTankArcade(0, 0);
664     distanceMove(15, -0.5);
665     // drive->runTankArcade(0, 0);

```

```

666 intake->run(true, false, -180); //start intake
667 pidTurn(0_deg, {0.007, 0.000008, 0});
668
669
670 //first neutral and to goal
671 // distanceMove(15, -0.8);
672 // distancePID(-15, {0.01, 0.0000008, 0});
673 // pidTurn(0_deg, {0.006, 0.000008, 0});
674 distanceMove(34, -0.6);
675 fourbarpneum->turnOn(); //clamp
676 pros::delay(100);
677 fourbar1->moveTarget(2400);
678 pidTurn(331_deg, {0.020, 0.000009, 0}); //turn to seesaw
679 distanceMove(51, -0.3); //move to seesaw
680 // distancePID(-27, {0.01, 0.0000008, 0});
681 pros::delay(300);
682 fourbar1->moveTarget(1900);
683 pros::delay(500);
684 fourbarpneum->turnOff();
685 pros::delay(200);
686 // fourbar1->moveTarget(2400);
687
688 //alliance currently in two bar
689 distanceMove(18, 0.4);
690 fourbar1->moveTarget(0);
691 backclamppneum->turnOff();
692 //move forwards and turn 180
693 pros::delay(1000);
694 distanceMove(9, -0.4);
695 double curr = imu.get_heading();
696 pidTurn((curr+175) * 1_deg, {0.005, 0.000008, 0});
697 //move forwards and clamp on goal
698 distanceMove(20, -0.4);
699 fourbarpneum->turnOn();
700 pros::delay(100);
701 //raise four bar
702 fourbar1->moveTarget(2100);
703 //turn back towards seesaw
704 pidTurn(345_deg, {0.006, 0.000008, 0});
705 //forward to seesaw
706 distanceMove(30, -0.6);
707 //drop goal
708 fourbar1->moveTarget(1900);
709 pros::delay(700);
710 fourbarpneum->turnOff();
711 pros::delay(200);
712
713 //alliance under first seesaw
714 printf("Moving back\n");
715 distanceMove(12, 0.4); //move back from seesaw
716 fourbar1->moveTarget(0); //lower four bar
717 pidTurn(272_deg, {0.009, 0.000008, 0}); //turn to the wall
718 //distanceMove(33, 0.4); //forward
719 drive->runTankArcade(0.4, 0); //run into wall
720 pros::delay(3500);
721 drive->runTankArcade(0, 0);
722 distanceMove(1, -0.5);

```

```

723 pidTurn(318_deg, {0.011, 0.000008, 0}); //turn towards goal under seesaw
724 distanceMove(31, -0.4); //forwards to that goal
725 fourbarpneum->turnOn(); //clamp
726 pros::delay(200);
727 distanceMove(20, 0.4); //get back out
728 fourbar1->moveTarget(2400); //raise four bar
729 pidTurn(226_deg, {0.010, 0.00001, 0}); //turn towards other side seesaw
730 distanceMove(84, -0.6); //beeline there
731 pros::delay(400);
732 fourbar1->moveTarget(1400); //lower four bar
733 pros::delay(1000);
734 fourbarpneum->turnOff(); //drop clamp
735
736 fourbar1->moveTarget(2400); //raise four bar
737
738 //tall neutral
739 distanceMove(13, 0.6); //back from seesaw
740 fourbar1->moveTarget(0);
741 pidTurn(305_deg, {0.014, 0.000008, 0}); //turn towards tall neutral
742 distanceMove(30, -0.6); //beeline to tall
743 fourbarpneum->turnOn();
744 pros::delay(400);
745 fourbar1->moveTarget(300);
746 distanceMove(50, -0.6); //beeline to tall
747 fourbarpneum->turnOff(); // drop tall
748 pros::delay(500);
749 distanceMove(9, 0.6);
750
751 //side alliance with back clamp
752 pidTurn(89_deg, {0.006, 0.000008, 0}); //turn towards side alliance
753 drive->runTankArcade(0.5, 0);
754 pros::delay(1700);
755 backclamppneum->turnOn();
756 drive->runTankArcade(0, 0);
757 distanceMove(15, -0.5);
758 fourbar1->moveTarget(0);
759 pidTurn(180_deg, {0.007, 0.000008, 0});
760
761 //last neutral goal
762 distanceMove(38, -0.8);
763 fourbarpneum->turnOn(); //clamp
764 pros::delay(100);
765 fourbar1->moveTarget(2400);
766 pidTurn(146_deg, {0.01, 0.000009, 0}); //turn to seesaw
767 distanceMove(40, -0.5); //move to seesaw
768 pros::delay(300);
769 fourbar1->moveTarget(2100); //drop four bar
770 pros::delay(500);
771 fourbarpneum->turnOff(); //drop neutral goal
772 pros::delay(200);
773 fourbar1->moveTarget(2400); //four bar back up
774
775 //deposit alliance on seesaw
776 distanceMove(18, 0.4); // away from seesaw
777 fourbar1->moveTarget(0); //lower four bar
778 backclamppneum->turnOff(); //let go of alliance goal
779 //move forwards and turn 180

```

```

780 distanceMove(8, -0.4); //go forward to lose the goal
781 curr = imu.get_heading();
782 pidTurn((curr+170)*1_deg, {0.008, 0.000008, 0}); //turn back towards the goal
783 distanceMove(13, -0.4); //move forwards and clamp on goal
784 fourbarpneum->turnOn();
785 pros::delay(100);
786 //raise four bar
787 fourbar1->moveTarget(2100);
788 //turn back towards seesaw
789 pidTurn(155_deg, {0.009, 0.000008, 0});
790 //forward to seesaw
791 distanceMove(40, -0.6);
792 //drop goal
793 fourbarpneum->turnOff();
794 distanceMove(15, 0.6);
795 pros::delay(200);
796 }
797
798 void skills() {
799     setEffectorPositions();
800     printf("done\n");
801     effectors.runOne(GOAL_LIFT, 1); //lower two-bar
802
803     distanceMove(43, -1); //move towards side neutral at full speed
804     fourbarpneum->turnOn(); //clamp it
805     pros::delay(300); //delay 300 ms
806     printf("Finished\n");
807     distanceMove(9, 1); //move back
808
809
810
811     OdomState goal = drive->getState();
812     goal.theta = 310_deg; //turn backside towards alliance goal
813     pidMoveTank(goal, {0, 0, 0}, turnDefault, true); //make the turn
814
815     pros::delay(300);
816     distanceMove(12, 1); //move towards alliance goal
817     effectors.runOne(GOAL_LIFT, 0);
818
819
820     goal = drive->getState();
821     goal.theta = 180_deg; //
822     pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true); //turn to dump goal
823     intake->run(true, false, -200); //start intake
824     fourbarpneum->turnOff(); //dump goal
825     pros::delay(750); //wait
826     goal = drive->getState();
827     goal.theta = 295_deg;
828     pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true); //turn towards center goal
829     pros::delay(200);
830     intake->run(true, false, 0); //end intake
831     fourbar1->moveTarget(0); //lower four bar
832     fourbar2->moveTarget(0); //lower four bar
833     distanceMove(54, -1); //move towards center goal
834     fourbarpneum->turnOn(); //clamp
835     pros::delay(300); //wait
836     //goal.theta = 225_deg;

```



```

837 //pidMoveTank(goal, {0, 0, 0}, {0.02, 0.000005, 0}, true); //turn slightly to be able to
get back into home zone
838 distanceMove(30, -1); //move goal forwards
839 // goal.theta = 225_deg;
840 // pidMoveTank(goal, {0, 0, 0}, {0.02, 0.000005, 0}, true); //drop neutral
841 fourbarpneum->turnOff();
842 distanceMove(5, 1);
843 goal.theta = 225_deg;
844 pidMoveTank(goal, {0, 0, 0}, {0.02, 0.000005, 0}, true); //turn towards other neutral
845 distanceMove(30, -1);
846 fourbarpneum->turnOn();
847 pros::delay(600);
848 // goal.theta = 0_deg;
849 // pidMoveTank(goal, {0, 0, 0}, {0.02, 0.000005, 0}, true); //turn towards other zone
850 distanceMove(35, 1);
851 }
852
853 void newSkills() {
854     setEffectorPositions();
855     effectors.runOne(GOAL_LIFT, 1); //two bar down
856     pros::delay(750); //wait
857     distanceMove(16, 0.7); //Move backwards to grab alliance goal
858     effectors.runOne(GOAL_LIFT, 0); //two bar up
859     distanceMove(2, -0.4); //moving away from seesaw
860     OdomState goal = drive->getState();
861     goal.theta = 270_deg;
862     pidMoveTank(goal, {0, 0, 0}, {0.007, 0.000005, 0}, true); //turn to 90 deg
863 //-----
864 //FIRST NEUTRAL
865
866     distanceMove(24, 1); //moving towards goal
867     goal.theta = 90_deg; //stop and turn
868     pidMoveTank(goal, {0, 0, 0}, {0.007, 0.000005, 0}, true); //turn 180 to face goal with
four bar
869     distanceMove(30, -1); //move the rest of the distance to the goal
870     fourbarpneum->turnOn(); //clamp
871     fourbar1->moveTarget(500); //raise four bar
872     fourbar2->moveTarget(500); //raise four bar
873     intake->run(true, false, -150); //start intake
874     distanceMove(30, -0.9); //move the rest of the way towards the ring line
875 //-----
876 //RINGS ON ALLIANCE GOAL
877     goal.theta = 180_deg; //make the turn towards 180 deg
878     pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true);
879     distanceMove(24, -1); //move quickly to the rings
880     distanceMove(48, -0.5); //then move slowly to intake the rings
881 //-----
882 //FIRST NEUTRAL ON PLATFORM
883
884     distanceMove(36, 0.8); //backwards to orient for the platform
885     goal.theta = 90_deg; //turn towards platform
886     pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true);
887     fourbar1->moveTarget(2400); //four bar up
888     fourbar2->moveTarget(2400);
889     pros::delay(300);
890     distanceMove(12, -0.4); //move towards platform
891     fourbar1->moveTarget(2000);
892     fourbar2->moveTarget(2000);

```



```

893     fourbarpneum->turnOff();           //release clamp
894     distanceMove(12, 0.4);             //back away from seesaw
895     fourbar1->moveTarget(0);           //four bar down
896     fourbar2->moveTarget(0);
897
898 // -----
899 // ALLIANCE GOAL MANIPULATION
900
901     goal.theta = 0_deg;                //turn towards forwards
902     pidMoveTank(goal, {0, 0, 0}, {0.007, 0.000005, 0}, true);
903     effectors.runOne(GOAL_LIFT, 1);    //drop two bar
904     distanceMove(12, 0.8);             //move forwards
905     effectors.runOne(GOAL_LIFT, 2);    //bring two bar all the way up
906     pros::delay(1000);                 //wait for that to happen before turning
907     goal.theta = 180_deg;              //turn around
908     pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true);
909     distanceMove(12, -1);              //Move forwards back towards the alliance goal
910     fourbarpneum->turnOn();            //clamp
911     fourbar1->moveTarget(2400);         //four bar up
912     fourbar2->moveTarget(2400);
913     pros::delay(1000);
914     goal.theta = 90_deg;               //turn towards platform
915     pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true);
916     fourbar1->moveTarget(2000);         //four bar slightly down
917     fourbar2->moveTarget(2000);
918     distanceMove(12, -0.4);            //move towards platform
919     fourbarpneum->turnOff();            //release clamp
920     distanceMove(12, 0.4);             //backwards movement
921     fourbar1->moveTarget(0);           //four bar down
922     fourbar2->moveTarget(0);
923
924 //-----
925 // SECOND SIDE NEUTRAL MANIPULATION
926
927     goal.theta = 180_deg;
928     pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true);
929     distanceMove(36, -0.9);            //moving towards the side neutral
930     goal.theta = 270_deg;              //turn towards it
931     pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true);
932     distanceMove(36, -1);              //move quickly towards it
933     fourbarpneum->turnOn();            //clamp
934     fourbar1->moveTarget(500);          //four bar up
935     fourbar1->moveTarget(500);
936     distanceMove(36, 0.9);             //move backwards to retrace steps
937     goal.theta = 0_deg;                //turn to 0
938     pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true);
939     fourbar1->moveTarget(2100);
940     fourbar2->moveTarget(2100);
941     distanceMove(36, -1);              //moving forwards to platform
942     goal.theta = 90_deg;
943     pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true);
944     distanceMove(12, -0.4);
945     fourbarpneum->turnOff();
946     distanceMove(12, 0.4);             //backwards from seesaw
947     fourbar1->moveTarget(0);           //four bar down
948     fourbar2->moveTarget(0);
949

```

```

950 //-----
951 //MOVEMENT FOR FIRST RED ALLIANCE GOAL
952
953     goal.theta = 180_deg;           //turning towards back wall
954     pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true);
955     distanceMove(48, -1);           //moving towards the turn location
956     goal.theta = 48_deg;           //turning towards alliance goal
957     pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true);
958     distanceMove(30, -0.6);         //move towards the goal
959     fourbarpneum->turnOn();         //clamp
960     distanceMove(30, 0.6);          //move away from the goal
961     fourbar1->moveTarget(500);      //four bar slightly up
962     fourbar2->moveTarget(500);      //four bar slightly up
963     pros::delay(500);
964     goal.theta = 0_deg;             //turning towards other alliance goal
965     pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true);
966 //-----
967 //MOVEMENT FOR SECOND RED ALLIANCE GOAL
968
969     distanceMove(80, -1);
970     effectors.runOne(GOAL_LIFT, 1); //lower two bar
971     pros::delay(1500);
972     distanceMove(15, -0.7);         //move the rest of the distance
973     pros::delay(500);
974     effectors.runOne(GOAL_LIFT, 0); // two bar up
975     distanceMove(1, 0.8);
976     goal.theta = 270_deg;
977     pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true);
978
979 //-----
980 //RINGS
981     fourbar1->moveTarget(2400);
982     fourbar2->moveTarget(2400);
983     distanceMove(96, -0.7);
984     goal.theta = 180_deg;
985     pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true);
986     distanceMove(3, -0.6);
987     fourbar1->moveTarget(100);
988     fourbar2->moveTarget(100);
989     autobalancer(23.5);
990 }
991
992 void leftskills() {
993     OdomState goal = drive->getState();
994     setEffectorPositions();
995     effectors.runOne(GOAL_LIFT, 1);           //lower goal lift
996     pros::delay(1500);
997     distanceMove(15, 0.5);                     // move forwards and get goal
998     effectors.runOne(GOAL_LIFT, 0);           // raise goal lift
999     pros::delay(500);
1000     goal = drive->getState();
1001     goal.theta = 270_deg;
1002     pidMoveTank(goal, {0, 0, 0}, {0.006, 0.000005, 0}, true); //turn to 90 deg
1003     distanceMove(24, 0.7);                     //move towards side neutral mogo
1004     pros::delay(500);
1005     goal = drive->getState();
1006     goal.theta = 110_deg;

```

```

1007 pidMoveTank(goal, {0, 0, 0}, {0.005, 0.000005, 0}, true); //turn to 90 deg
1008 distanceMove(32, -1); //move towards side neutral mogo
1009
1010 fourbarpneum->turnOn(); //clamp
1011 pros::delay(300);
1012 fourbar1->moveTarget(2400); //four bar up
1013 fourbar2->moveTarget(2400);
1014
1015 goal = drive->getState();
1016 goal.theta = 90_deg; //turn towards center
1017 pidMoveTank(goal, {0, 0, 0}, {0.01, 0.000005, 0}, true);
1018 distanceMove(10, -0.5); //move towards ring cross line
1019 goal = drive->getState();
1020 goal.theta = 180_deg; //turn towards ring crosses
1021 pidMoveTank(goal, {0, 0, 0}, {0.008, 0.000005, 0}, true);
1022 intake->run(true, false, -150);
1023 // effectors.runOne(GOAL_LIFT, 1);
1024 distanceMove(40, -0.5); //move through rings
1025 distanceMove(3, 0.8); //move back to platform
1026 goal = drive->getState();
1027 goal.theta = 92_deg; //turn towards platform
1028 pidMoveTank(goal, {0, 0, 0}, {0.008, 0.000005, 0}, true);
1029 distanceMove(14, -0.5);
1030 fourbar1->moveTarget(1800); //four bar down to balance platform
1031 fourbar2->moveTarget(1800);
1032 pros::delay(500);
1033 fourbarpneum->turnOff(); //release clamp
1034 pros::delay(500);
1035 fourbar1->moveTarget(2400);
1036 fourbar2->moveTarget(2400);
1037 pros::delay(500);
1038 distanceMove(2, 0.5); //move away from platform
1039 goal = drive->getState();
1040 goal.theta = 180_deg; //turn towards other side neutral
1041 pidMoveTank(goal, {0, 0, 0}, {0.007, 0.000005, 0}, true);
1042 intake->run(true, false, -150);
1043 fourbar1->moveTarget(0); //four bar down
1044 fourbar2->moveTarget(0);
1045 distanceMove(35, -0.65); //move towards side neutral
1046 goal = drive->getState();
1047 goal.theta = 272_deg; //turn towards other side neutral
1048 pidMoveTank(goal, {0, 0, 0}, {0.008, 0.000005, 0}, true);
1049 distanceMove(35, -0.65);
1050 fourbarpneum->turnOn(); //clamp
1051
1052 distanceMove(30, 0.65); //move backwards to line
1053 fourbar1->moveTarget(2400);
1054 fourbar2->moveTarget(2400);
1055 goal = drive->getState();
1056 goal.theta = 359_deg;
1057 pidMoveTank(goal, {0, 0, 0}, {0.008, 0.000005, 0}, true);
1058 distanceMove(36, -0.65); //move backwards to goal
1059 pros::delay(500);
1060 fourbar1->moveTarget(2150); //lower four bar
1061 fourbar2->moveTarget(2150);
1062 goal = drive->getState();
1063 goal.theta = 86_deg; //turn towards platform

```

```

1064 pidMoveTank(goal, {0, 0, 0}, {0.0085, 0.000008, 0}, true);
1065 distanceMove(4, -0.8); //move forwards to platform
1066
1067 fourbar1->moveTarget(2000); //lower four bar
1068 fourbar2->moveTarget(2000);
1069 pros::delay(750);
1070
1071 fourbarpneum->turnOff(); //release clamp
1072 pros::delay(500);
1073 fourbar1->moveTarget(2200); //lower four bar
1074 fourbar2->moveTarget(2200);
1075
1076 pros::delay(500);
1077 distanceMove(3, 0.6); //backwards from platform
1078
1079
1080 goal = drive->getState();
1081 goal.theta = 180_deg;
1082 pidMoveTank(goal, {0, 0, 0}, {0.007, 0.00000, 0}, true);
1083 effectors.runOne(GOAL_LIFT, 1); //drop two bar
1084 fourbar1->moveTarget(0); //four bar down
1085 fourbar2->moveTarget(0);
1086 distanceMove(8, -0.6); //move forwards
1087 pros::delay(500);
1088 effectors.runOne(GOAL_LIFT, 2); //two bar up
1089 goal.theta = (imu.get_heading() + 180) * 1_deg; //turn around to go
back towards the dropped alliance goal
1090 pidMoveTank(goal, {0, 0, 0}, {0.005, 0.00000, 0}, true);
1091 distanceMove(14, -0.8); //move towards alliance goal
1092 pros::delay(500);
1093 fourbarpneum->turnOn(); //clamp
1094 pros::delay(500);
1095 fourbar1->moveTarget(2400); //four bar up
1096 fourbar2->moveTarget(2400);
1097 distanceMove(6, 0.8); //move towards alliance goal
1098 goal = drive->getState();
1099 goal.theta = 87_deg; //turn back towards platform with alliance
goal
1100 pidMoveTank(goal, {0, 0, 0}, {0.008, 0.000008, 0}, true);
1101 // fourbar1->moveTarget(2400); //four bar up
1102 // fourbar2->moveTarget(2400);
1103 pros::delay(500);
1104 fourbar1->moveTarget(2000);
1105 fourbar2->moveTarget(2000);
1106 distanceMove(4, -0.6); //forwards to platform
1107 // fourbar1->moveTarget(2000); //four bar down to deposit
1108 // fourbar2->moveTarget(2000);
1109 fourbarpneum->turnOff(); //release clamp
1110 pros::delay(500);
1111 distanceMove(3, 0.8); //backwards from platform
1112 goal.theta = 170_deg; //turn towards alliance goal that fell off
the seesaw
1113 pidMoveTank(goal, {0, 0, 0}, {0.007, 0.000008, 0}, true);
1114 fourbar1->moveTarget(0); //lower four bar
1115 fourbar2->moveTarget(0);
1116 distanceMove(37, -1); //moving forwards towards the first alliance
goal that came off the seesaw
1117 goal.theta = 50_deg; //turn towards alliance goal

```

```

1118 pros::delay(500);
1119 pidMoveTank(goal, {0, 0, 0}, {0.007, 0.000008, 0}, true);
1120 distanceMove(28, -0.5); //move towards alliance goal
1121 fourbarpneum->turnOn(); //clamp
1122 distanceMove(24, 0.7); //Move away from alliance goal
1123 fourbar1->moveTarget(2400); //four bar slightly up
1124 fourbar2->moveTarget(2400);
1125 // goal.theta = 85_deg;
1126 // pidMoveTank(goal, {0, 0, 0}, {0.007, 0.000008, 0}, true);
1127 // distanceMove(6, 1);
1128 // pros::delay(250);
1129 goal.theta = 141_deg; //turn so that we can move backwards to the
other alliance goal
1130 pidMoveTank(goal, {0, 0, 0}, {0.005, 0.000008, 0}, true);
1131 effectors.runOne(GOAL_LIFT, 1); //drop two bar
1132 pros::delay(500);
1133 distanceMove(110, 0.8); //move quickly to the alliance goal
1134 pros::delay(750);
1135 effectors.runOne(GOAL_LIFT, 0); //raise two bar
1136 pros::delay(1000);
1137 goal.theta = 270_deg; //turn to face the rings
1138 pidMoveTank(goal, {0, 0, 0}, {0.007, 0.000008, 0}, true);
1139 fourbar1->moveTarget(2400); //four bar up a bit
1140 fourbar2->moveTarget(2400);
1141 intake->run(true, false, -150); //start intake
1142
1143
1144 speedMove(1000, -0.7);
1145 distanceMove(1, 0.5);
1146 //distanceMove(86, -0.6); //move along the rings
1147 goal.theta = 175_deg; //turn to climb on the seesaw
1148 pidMoveTank(goal, {0, 0, 0}, {0.007, 0.000008, 0}, true);
1149 // fourbar1->moveTarget(2000); //four bar up to put the seesaw down
1150 // fourbar2->moveTarget(2000);
1151 distanceMove(1, -0.5);
1152 fourbar1->moveTarget(0); //four bar down to put the seesaw down
1153 fourbar2->moveTarget(0);
1154 pros::delay(2000);
1155 fourbar1->moveTarget(150);
1156 fourbar2->moveTarget(150);
1157 // distanceMove(40, -1.0); //autobalance
1158 autobalancer(24);
1159 pros::delay(10000);
1160 }
1161
1162 void rightrings() {
1163     setEffectorPositions();
1164     fourbar1->moveTarget(500); //lift four bar to intake
1165     fourbar2->moveTarget(500);
1166     effectors.runOne(GOAL_LIFT, 1); //lower two bar
1167     pros::delay(1500);
1168     distanceMove(20, 1); //pick up alliance
1169     effectors.runOne(GOAL_LIFT, 0); //raise two bar
1170     distanceMove(12, -1); //move back
1171     intake->run(true, false, -175); //turn on intake
1172     while(1) {
1173         //oscillate to pick up rings

```

```

1174     distanceMove(7, -0.5);
1175     pros::delay(500);
1176     distanceMove(7, 0.5);
1177 }
1178
1179
1180 }
1181
1182 void rightMiddle() {
1183     distanceMove(53, -1); //move towards side neutral at full speed
1184     fourbarpneum->turnOn(); //clamp it
1185     pros::delay(300); //delay 300 ms
1186     printf("Finished\n");
1187     distanceMove(53, 1); //move back
1188 }
1189
1190 void left() {
1191     setEffectorPositions();
1192     printf("done\n");
1193     distanceMove(43, -1); //move towards side neutral at full speed
1194     fourbarpneum->turnOn(); //clamp it
1195     pros::delay(200);
1196     printf("Finished\n");
1197     distanceMove(10, 1); //move back
1198     pidTurn(20_deg, {0.02, 0.000008, 0});
1199     distanceMove(7, 0.6);
1200 }
1201
1202 void middle() {
1203     setEffectorPositions();
1204     printf("done\n");
1205     distanceMove(50, -1); //move towards side neutral at full speed
1206     fourbarpneum->turnOn(); //clamp it
1207     pros::delay(100);
1208     printf("Finished\n");
1209     distanceMove(10, 1); //move back
1210     distanceMove(14, 0.6);
1211 }
1212
1213 void rightthenmiddle() {
1214     setEffectorPositions();
1215     printf("done\n");
1216     distanceMove(39, -1); //move towards side neutral at full speed
1217     fourbarpneum->turnOn(); //clamp it
1218     pros::delay(100);
1219     printf("Finished\n");
1220     distanceMove(10, 1); //move back
1221     pidTurn(270_deg, {0.01, 0.000008, 0});
1222     fourbarpneum->turnOff();
1223 }
1224
1225
1226 void leftmiddle() {
1227     setEffectorPositions();
1228     intake->run(true, false, -180);
1229     printf("done\n");
1230     distanceMove(69, -0.7); //move towards side neutral at full speed

```



```

1231     fourbarpneum->turnOn(); //clamp it
1232     pros::delay(100);
1233     printf("Finished\n");
1234     distanceMove(10, 1); //move back
1235     distanceMove(20, 0.6);
1236 }
1237
1238
1239
1240 void esbensOdom() {
1241     // jank, Esben-coded odom that involves taking the current angle at the middle of an
interval of 50 ms and after 50 ms,
1242     // calculating a linear distance in x and y even if the movement is a curve
1243     // essentially a linear approximation of the movement, 20 times per second
1244
1245     float x = 0;          //variables for tracking
1246     float y = 0;
1247
1248     ADIEncoder righttrack = ADIEncoder('A', 'B', false);          //encoders because i don't
know how to get values
1249     ADIEncoder lefttrack = ADIEncoder('C', 'D', true);
1250
1251     righttrack.reset(); //reset encoders to zero
1252     lefttrack.reset();
1253
1254     float currangle = imu.get_heading();
1255     float right = 0;
1256     float left = 0;
1257
1258     while (true)
1259     {
1260         pros::delay(25);
1261         currangle = imu.get_heading();
1262         pros::delay(25);
1263         int currright = righttrack.get();
1264         int currleft = lefttrack.get();
1265         int rightencchange = currright - right;
1266         int leftencchange = currleft - left;
1267         float average = (rightencchange + leftencchange)/2 / 360 * 2.75 * PI;
1268         x += average * cos(currangle);
1269         y += average * sin(currangle);
1270         pros::c::lcd_print(0, "OdomX: %f\n", x);          //display on lcd screen
1271         pros::c::lcd_print(1, "OdomY: %f\n", y);
1272         pros::c::lcd_print(2, "OdomH: %d\n", currangle);
1273         right = currright;
1274         left = currleft;
1275     }
1276
1277 }
1278
1279
1280
1281
1282
1283 void autonomous() {
1284
1285     //okapi::Controller controller (okapi::ControllerId::master);
1286     drive->setMode(okapi::AbstractMotor::brakeMode::hold);

```

```

1287     left();
1288     //right();
1289     //middle();
1290     //left();
1291     //leftfast();
1292     //drive->setMode(okapi::AbstractMotor::brakeMode::coast);
1293 }
1294
1295 //experimental pure pursuit handler
1296 void PurePursuitHandler() {
1297     /*
1298     std::vector<point> points;
1299     points.push_back({0, 0, 0, 0, 0});
1300     points.push_back({0, 24, 0, 0, 0});
1301     points.push_back({-15, 40, 0, 0, 0});
1302     PurePursuitPathGen path = PurePursuitPathGen(3, 0.25, 0.75,0.001, points,10.0, 10.0, 2);
1303     path.interpolate();
1304     path.smooth();
1305     path.calc_distances();
1306     path.calc_curvature();
1307     path.calc_velocities();
1308     path.print_path();
1309     PurePursuitFollower follow = PurePursuitFollower(8);
1310     follow.read(path);
1311     std::array<double, 4> vels = {0, 0, 0, 0};
1312     double x, y, theta;
1313     do {
1314         theta = 90-imu.get_heading();
1315         x = drive->getX();
1316         y = drive->getY();
1317         vels = follow.follow(y, x, theta);
1318         printf("POS: %f %f %f\n", y, x, theta);
1319         printf("%f %f %f %f\n", vels[0], vels[1], vels[2], vels[3]);
1320         drive->runTankArcade(vels[0], vels[1]);
1321         pros::delay(30);
1322     } while(vels[0] != 0 && vels[1] != 0 &&vels[2] != 0 &&vels[3] != 0);
1323     drive->runTank(0, 0);
1324 */
1325 }
1326 }

```



```
1 #include "main.h"
2 #include "pid.h"
3 #include "drive.h"
4
5 //testing push
```

```

1 #include "button.h"
2
3 Button::Button() {
4     But but;
5     for(okapi::ControllerDigital x : buttonList) {
6         but = {false, 0};
7         std::pair<okapi::ControllerDigital , But> myBut (x,but);
8         buttons.insert(myBut);
9     }
10 }
11
12 //handle button counts and states
13 void Button::handleButtons(Controller controller) {
14     for (auto& [key, value]: buttons) {
15         if(controller.getDigital(key) && !value.state) {
16             value.state = true;
17             value.count++;
18         }
19         else if(!controller.getDigital(key) && value.state) {
20             value.state = false;
21         }
22     }
23 }
24
25 //return counts
26 int Button::getCount(okapi::ControllerDigital id) {
27     return buttons[id].count;
28 }
29
30 //return states
31 bool Button::getPressed(okapi::ControllerDigital id) {
32     return buttons[id].state;
33 }

```

```

1 #include "drive.h"
2
3
4 //creates okapi chassis object
5 Drive::Drive() {
6     chassis = ChassisControllerBuilder()
7         .withMotors(
8             {-TOP_LEFT_MOTOR, -LEFT_MIDDLE_MOTOR, -BOTTOM_LEFT_MOTOR},
9             {TOP_RIGHT_MOTOR, RIGHT_MIDDLE_MOTOR, BOTTOM_RIGHT_MOTOR})
10        .withDimensions(
11            AbstractMotor::gearset::green,
12            ChassisScales({WHEELDIM, WHEELTRACK}, imev5GreenTPR))
13        .withSensors(
14            ADIEncoder( // left encoder
15                LEFT_TRACKING_WHEEL_TOP,
16                LEFT_TRACKING_WHEEL_BOTTOM
17            ),
18            ADIEncoder( // right encoder
19                RIGHT_TRACKING_WHEEL_TOP,
20                RIGHT_TRACKING_WHEEL_BOTTOM,
21                true
22            )
23        )
24    )
25    .withOdometry(
26        ChassisScales({ODOMWHEELDIM, ODOMTRACK}, quadEncoderTPR)
27    )
28
29    .buildOdometry();
30    speedfactor = 1;
31 }
32
33
34
35
36 //returns X of odometry
37 double Drive::getX() {
38     return chassis->getState().x.convert(inch);
39 }
40
41 //returns Y of odometry
42 double Drive::getY() {
43     return chassis->getState().y.convert(inch);
44 }
45
46 //returns odometry heading
47 double Drive::getHeading() {
48     return chassis->getState().theta.convert(degree);
49 }
50
51
52 //arcade move function for X drive (old)
53 void Drive::run(double forward, double strafe, double heading) {
54     std::shared_ptr<okapi::XDriveModel> xDrive = std::static_pointer_cast<okapi::XDriveModel>
55     (chassis->getModel());
56     if(forward+strafe+heading>1) {
57         forward/=(forward+strafe+heading);

```

```

57     strafe/=(forward+strafe+heading);
58     heading/=(forward+strafe+heading);
59 }
60 printf("%f %f %f\n", strafe, forward, heading);
61 xDrive->xArcade(strafe, forward, heading);
62 }
63
64 //arcade move function for tank drive
65 void Drive::runTankArcade(double forward, double turn) {
66     chassis->getModel()->arcade(forward, turn);
67 }
68
69 //tank move function for tank drive
70 void Drive::runTank(double left, double right) {
71     chassis->getModel()->tank(left, right);
72 }
73
74
75 //returns all of odometry state (x, y, and theta)
76 okapi::OdomState Drive::getState() {
77     return chassis->getState();
78 }
79
80 //reverses orientation for driver
81 void Drive::reverseOrientation(int ori) {
82     if(ori%2 == 1) {
83         printf("REVERSED\n");
84         speedfactor = -1;
85     }
86     else {
87         speedfactor = 1;
88     }
89 }
90
91 //sets brake mode of drive mode (if need to coast or hold)
92 void Drive::setMode(okapi::AbstractMotor::brakeMode brakeMode) {
93     chassis->getModel()->setBrakeMode(brakeMode);
94 }
95
96 // double Drive::getEncoder() {
97 //     return enc.get();
98 // }

```

```

1 #include "effectors.h"
2
3 //reset encoders for effectors
4 Effectors::Effectors() {
5     for(int i = 0; i < 1; i++) {
6         motors[i].getEncoder().reset();
7     }
8 }
9
10 //set all encoder positions for two bar
11 void Effectors::addPosition() {
12     //900 difference between upper and lower position
13     encPositions[0][0] = 1530; // Two bar upper position
14     encPositions[0][1] = 2350; // Two bar lower position
15     encPositions[0][2] = 0;     //two bar starting position
16     prevCounts[0] = 0;
17     prevCounts[1] = 0;
18     prevCounts[2] = 0;
19
20 }
21
22 //handle two bar in opcontrol
23 void Effectors::step(int buttons[3], double speeds[3]) {
24
25     buttons[0] = buttons[0] % 2;
26     buttons[1] = buttons[1] % 2;
27
28     for(int i = 0; i < 1; i++) {
29         //printf("Enc position: %f", motors[i].getPosition());
30         if(buttons[i] != prevCounts[i]) {
31             motors[i].moveAbsolute(encPositions[i][buttons[i]], speeds[i]);
32         }
33
34     }
35     for(int i = 0; i < 1; i++) {
36         prevCounts[i] = buttons[i];
37     }
38 }
39
40 //move two bar to preset position
41 void Effectors::runOne(int lift, int pos) {
42     motors[lift].moveAbsolute(encPositions[lift][pos], 200);
43 }
44
45 //move two bar to any position
46 void Effectors::runOneToPosition(int lift, int pos) {
47     motors[lift].moveAbsolute(pos, 200);
48 }

```

```

1 #include "intake.h"
2
3 Intake::Intake(double port) : m(port)
4 {
5     m.setBrakeMode(okapi::AbstractMotor::brakeMode::hold);
6     m.getEncoder().reset();
7 }
8
9 void Intake::addPosition(int pos) {
10     encPositions.push_back(pos);
11 }
12
13
14 //move four bar at full speed to position
15 void Intake::moveTarget(double enc) {
16     m.moveAbsolute(enc, 200);
17 }
18
19
20 //set limits of four bar
21 void Intake::setLimits(int upper, int lower) {
22     this->upper = upper;
23     this->lower = lower;
24     limits = true;
25 }
26
27
28 //run intake at speed while obeying limits
29 void Intake::run(bool left, bool right, double speed) {
30     if(limits && ((m.getPosition()>upper && left) || (m.getPosition()<lower && right))) {
31         m.moveVelocity(0);
32     }
33     else if(left) {
34         m.moveVelocity(speed);
35     }
36     else if(right) {
37         m.moveVelocity(-speed);
38     }
39     else if(!left && !right) {
40         m.moveVelocity(0);
41     }
42 }
43
44 void Intake::handle(int count, double speed) {
45     if(count%2 == 1 && count!= prevCount) {
46         m.moveVelocity(-speed);
47     }
48     if(count%2 == 0 && count!= prevCount) {
49         m.moveVelocity(0);
50     }
51 }
52
53 void Intake::stepAbsolute(int count, double speed) {
54     printf("count: %d\n", count % encPositions.size());
55     if(prevCount != count) {
56         double target = encPositions[count % encPositions.size()];
57         m.moveAbsolute(target, speed);

```

```
58 | }  
59 | prevCount = count;  
60 | }
```

```

1 #include "odometry.h"
2
3 /*
4 Odometry::Odometry(ADIEncoder left, ADIEncoder right, ADIEncoder back) {
5     this->left = left;
6     this->right = right;
7     this->back = back;
8 }
9
10 OdomState Odometry::step() {
11     int left = left.get();
12     int right = right.get();
13     int back = back.get();
14
15     int leftchange = left-prevLeft;
16     int rightchange = right-prevRight;
17     int backchange = back-prevBack;
18
19     double leftDistance = (leftchange / 360) * PI * ODOMWHEELDIM;
20     double rightDistance = (rightchange / 360) * PI * ODOMWHEELDIM;
21     double backDistance = (backchange / 360) * PI * ODOMWHEELDIM;
22
23     prevLeft = left;
24     prevRight = right;
25     prevBack = back;
26
27     double currHeading = 90-imu.get_heading();
28
29     double anglediff = currHeading - prevHeading;
30     prevHeading = currHeading;
31
32     double localXOffset;
33     double localYOffset;
34
35     if (anglediff == 0_deg){
36         localXOffset = backDistance;
37         localYOffset = rightDistance;
38     }
39     else {
40         localXOffset = 2.0 * sin(angleDiff/2.0) * ( ( backDistance/angleDiff ) + backDistance );
41         localYOffset = 2.0 * sin(anglediff/2.0) * ( ( rightDistance/angleDiff ) + rightDistance );
42     }
43     double averageOrientation = currHeading + (deltaHeading/2.0);
44     double r = sqrt( (localXOffset * localXOffset) + (localYOffset * localYOffset) );
45     double theta = atan2(localYOffset , localXOffset);
46     theta *= (180 / PI);
47
48     theta -= averageOrientation;
49
50     OdomState currState = {prevState.x + (localXOffset*1_in), prevState.y + (localYOffset*1_in),
51 }
52 }
53 */

```



```
1 #include "pid.h"
2
3 //constructor sets constants
4
5 PID::PID(PIDConst constants) {
6     this->kp = constants.kp;
7     this->ki = constants.ki;
8     this->kd = constants.kd;
9 }
10
11 //steps PI algo with error passed in (no D)
12 double PID::step(double err) {
13     totalerr+=err;
14     double val = kp*err + totalerr*ki;
15     return val;
16 }
```

```

1 #include "pneumatics.h"
2
3 //constructor
4 Pneumatics::Pneumatics(uint8_t port) : piston(port) {
5     // turnOff();
6 }
7
8 //handle function for buttons
9 void Pneumatics::handle(int count) {
10     if(count%2 == 0 && count!= prevCount) {
11         turnOn();
12     }
13     if(count%2 == 1 && count!= prevCount) {
14         turnOff();
15     }
16 }
17
18
19 //actuates pneums down
20 void Pneumatics::turnOn() {
21     printf("pleaseturnon");
22     piston.set_value(4095);
23     state = true;
24 }
25
26 //actuates pneums up
27 void Pneumatics::turnOff() {
28     printf("isthisrunning\n");
29     piston.set_value(0);
30     printf("off\n");
31     state = false;
32 }
33
34 //testing function
35 void Pneumatics::onThenOff(int delay) {
36     turnOn();
37     pros::delay(1000);
38     printf("hi");
39     turnOff();
40 }
41
42 void Pneumatics::offThenOn(uint32_t delay) {
43     turnOff();
44     pros::delay(delay);
45     turnOn();
46 }

```

```

1 #include "../include/PurePursuitFollower.h"
2 #include "PurePursuitPathGen.h"
3
4 #include <vector>
5 #include <string>
6 #include <fstream>
7 #include <iostream>
8 #include <algorithm>
9 #include <math.h>
10 #include <array>
11
12 #define PI 3.14159265
13
14 PurePursuitFollower::PurePursuitFollower(double lookahead) {
15     this->lookahead = lookahead;
16     this->prev_time = timer.millis().convert(second);
17 }
18
19 void PurePursuitFollower::read_from_file(std::string filename) {
20     std::ifstream fin;
21     fin.open(filename);
22     points.clear();
23     followPoint temp;
24     while (!fin.eof()) {
25         fin >> temp.x >> temp.y >> temp.vel;
26         points.push_back(temp);
27     }
28 }
29
30 void PurePursuitFollower::read(PurePursuitPathGen obj) {
31     std::vector<point> temppoints;
32     temppoints = obj.get_points();
33     followPoint temp;
34     printf("READING\n");
35     for(int i = 0; i < temppoints.size(); i++) {
36         temp.x = temppoints[i].x;
37         temp.y = temppoints[i].y;
38         temp.vel = temppoints[i].vel;
39         points.push_back(temp);
40     }
41     for(followPoint x: points) {
42         printf("%f %f %f\n", x.x, x.y, x.vel);
43     }
44 }
45
46
47 void PurePursuitFollower::calc_closest_point(double x, double y) {
48     double min = 1E7;
49     for (int i = last_closest_point; i < points.size(); i++) {
50         double dist = sqrt(((x - points[i].x) * (x - points[i].x)) + ((y - points[i].y) * (y -
points[i].y)));
51         if (dist < min) {
52             min = dist;
53             last_closest_point = i;
54             closest_point = points[i];
55         }
56     }

```

```

57 }
58
59 void PurePursuitFollower::calc_lookahead(double x, double y) {
60     std::pair<double, double> d, f;
61     double a, b, c, discriminant, t1, t2;
62     for (int i = last_closest_point + 1; i < points.size(); i++) {
63         d.first = points[i].x - points[i - 1].x;
64         d.second = points[i].y - points[i - 1].y;
65         f.first = points[i].x - x;
66         f.second = points[i].y - y;
67         a = d.first * d.first + d.second * d.second;
68         b = 2 * (f.first * d.first + f.second * d.second);
69         c = (f.first * f.first + f.second * f.second) - lookahead * lookahead;
70         discriminant = b * b - (4 * a * c);
71         if (discriminant >= 0) {
72             discriminant = sqrt(discriminant);
73             t1 = (-b - discriminant) / (2 * a);
74             t2 = (-b + discriminant) / (2 * a);
75             if (t1 >= 0 && t1 <= 1 && t1 + i - 1 > last_fractional_index) {
76                 lookahead_point.first = points[i - 1].x + (t1 * d.first);
77                 lookahead_point.second = points[i - 1].y + (t1 * d.second);
78                 last_lookahead_point = lookahead_point;
79                 break;
80             }
81             if (t2 >= 0 && t2 <= 1 && t2 + i - 1 > last_fractional_index) {
82                 lookahead_point.first = points[i - 1].x + (t2 * d.first);
83                 lookahead_point.second = points[i - 1].y + (t2 * d.second);
84                 last_lookahead_point = lookahead_point;
85                 break;
86             }
87         }
88     }
89     lookahead_point = last_lookahead_point;
90     last_lookahead_point = lookahead_point;
91 }
92
93 void PurePursuitFollower::calc_curvature_at_point(double x, double y, double theta) {
94     double xtemp;
95     double a, b, c;
96     a = -tan((theta));
97     b = 1;
98     c = (tan((theta)) * x) - y;
99     double temp = (sin((theta)) * (lookahead_point.first - x)) - (cos((theta)) *
100 (lookahead_point.second - y));
101     int sign = (temp > 0) ? 1 : ((temp < 0) ? -1 : 0);
102     xtemp = abs((a * lookahead_point.first) + (b * lookahead_point.second) + c) / sqrt((a * a
103 + (b * b)));
104     this->curvature = ((2 * xtemp) / (lookahead * lookahead));
105     this->curvature *= sign;
106 }
107
108 std::array<double, 4> PurePursuitFollower::follow_sim(double x, double y, double theta) {
109     calc_closest_point(x, y);
110     calc_lookahead(x, y);
111     calc_curvature_at_point(x, y, theta);
112     std::array<double, 4> vels;
113     if (closest_point.x == points[points.size() - 1].x && closest_point.y ==
114 points[points.size() - 1].y) {

```

```

112         vels[0] = 0;
113         vels[1] = 0;
114         vels[2] = 0;
115         vels[3] = 0;
116         return vels;
117     }
118     vels[0] = closest_point.vel;
119     vels[1] = vels[0] * curvature;
120     vels[2] = 0;
121     vels[3] = 0;
122     return vels;
123 }
124
125 std::array<double, 4> PurePursuitFollower::follow(double x, double y, double theta) {
126     calc_closest_point(x, y);
127     calc_lookahead(x, y);
128     calc_curvature_at_point(x, y, theta);
129     std::array<double, 4> vels;
130     if (closest_point.x == points[points.size() - 1].x && closest_point.y ==
points[points.size() - 1].y) {
131         vels[0] = 0;
132         vels[1] = 0;
133         vels[2] = 0;
134         vels[3] = 0;
135         return vels;
136     }
137     double time = (timer.millis().convert(second)-prev_time);
138     double vel, ang;
139     vel = (closest_point.vel);
140     vel = prev_vel+(std::clamp(vel-prev_vel, -(time*max_accel), (time*max_accel)));
141     printf("vel: %f curvature: %f\n", vel, curvature);
142     vel = vel/10;
143     ang = vel*curvature;
144     ang = ang/(10/(2*PI));
145     vels[0] = vel;
146     vels[1] = ang;
147     vels[2] = vel;
148     vels[3] = ang;
149     prev_time = timer.millis().convert(second);
150     prev_vel = vel;
151     return vels;
152 }

```

```

1 #include "PurePursuitPathGen.h"
2 #include <vector>
3 #include <string>
4 #include <fstream>
5 #include <iostream>
6 #include <algorithm>
7 #include <math.h>
8
9 PurePursuitPathGen::PurePursuitPathGen(double spacing, double a, double b, double tolerance,
std::vector<point> points, double max_vel, double max_accel, int k) {
10     this->spacing = spacing;
11     this->a = a;
12     this->b = b;
13     this->tolerance = tolerance;
14     this->max_vel = max_vel;
15     this->max_accel = max_accel;
16     this->k = k;
17     this->initial_points = points;
18 }
19
20 void PurePursuitPathGen::interpolate() {
21     point vec;
22     int mag;
23     int num_points;
24     final_points.push_back(initial_points[0]);
25     for(int i = 1; i < initial_points.size(); i++) {
26
27
28         vec.x = initial_points[i].x-initial_points[i-1].x;
29         vec.y = initial_points[i].y-initial_points[i-1].y;
30
31         mag = sqrt((vec.x*vec.x)+(vec.y*vec.y));
32
33         num_points = ceil(mag/spacing);
34
35         vec.x = (vec.x/mag)*spacing;
36         vec.y = (vec.y/mag)*spacing;
37         point new_vec;
38         for(int j = 1; j < num_points; j++) {
39             new_vec.x = (initial_points[i-1].x+(vec.x*j));
40             new_vec.y = (initial_points[i-1].y+(vec.y*j));
41             final_points.push_back(new_vec);
42         }
43         final_points.push_back(initial_points[i]);
44     }
45 }
46 void PurePursuitPathGen::calc_distances() {
47     final_points[0].distance = 0;
48     for(int i = 1; i < final_points.size(); i++) {
49         final_points[i].distance = final_points[i-1].distance + sqrt(pow(final_points[i].x-
final_points[i-1].x, 2)+pow(final_points[i].y-final_points[i-1].y, 2));
50     }
51 }
52 void PurePursuitPathGen::calc_curvature() {
53     final_points[0].curve = 0;
54     double k1, k2, center1, center2, r, x1, x2, x3, y1, y2, y3;
55     for(int i = 1; i < final_points.size()-1; i++) {

```

```

56     x1 = final_points[i].x+0.001;
57     x2 = final_points[i-1].x;
58     x3 = final_points[i+1].x;
59
60     y1 = final_points[i].y;
61     y2 = final_points[i-1].y;
62     y3 = final_points[i+1].y;
63
64     k1=0.5*((x1*x1)+(y2*y2)-(x2*x2)-(y2*y2))/(x1-x2);
65     k2 = (y1-y2)/(x1-x2);
66     center2 = 0.5*((x2*x2)-(2*x2*k1)+(y2*y2)-(x3*x3)+(2*x3*k1)-(y3*y3))/(((x3*k2)-y3+y2-
(x2*k2)));
67     center1 = k1-k2*center2;
68     r = sqrt((x1-center1)*(x1-center1) + (y1-center1)*(y1-center1));
69     final_points[i].curve = 1/r;
70 }
71 final_points[final_points.size()-1].curve = 0;
72 }
73 void PurePursuitPathGen::smooth() {
74     std::vector<point> copy;
75     copy = final_points;
76     double change = tolerance;
77     while(change>=tolerance) {
78         change = 0.0;
79         for(int i = 1; i < final_points.size()-1; i++) {
80             double aux = copy[i].x;
81             copy[i].x += a*(final_points[i].x-copy[i].x) + b*(copy[i-1].x + copy[i+1].x-(2.0*
(copy[i].x)));
82             change+=abs(aux-copy[i].x);
83             aux = copy[i].y;
84             copy[i].y += a*(final_points[i].y-copy[i].y) + b*(copy[i-1].y + copy[i+1].y-(2.0*
(copy[i].y)));
85             change+=abs(aux-copy[i].y);
86         }
87     }
88     final_points = copy;
89 }
90
91 void PurePursuitPathGen::calc_velocities() {
92     for(int i = 0; i < final_points.size(); i++) {
93         //std::cout << "Max vels" << k/final_points[i].curve << "\n";
94         final_points[i].vel = std::min(max_vel, k/final_points[i].curve);
95     }
96     final_points[final_points.size()-1].vel = 0;
97     for(int i = final_points.size()-2; i >=0; i--) {
98         final_points[i].vel = std::min(final_points[i].vel, sqrt(pow(final_points[i+1].vel,
2)+2*max_accel*(final_points[i+1].distance-final_points[i].distance)));
99     }
100 }
101
102 void PurePursuitPathGen::write_to_file() {
103     std::ofstream fout;
104     fout.open("path.txt");
105     for(int i = 0; i < final_points.size(); i++) {
106         fout << final_points[i].x << " " << final_points[i].y << " " << final_points[i].vel <<
"\n";
107     }
108     fout.close();

```

```

109 }
110 void PurePursuitPathGen::print_path() {
111     printf("INITIAL\n");
112     for(int i = 0; i < initial_points.size(); i++) {
113         printf("%f %f\n", initial_points[i].x, initial_points[i].y);
114     }
115     printf("FINAL\n");
116     for(int i = 0; i < final_points.size(); i++) {
117         printf("%f %f\n", final_points[i].x, final_points[i].y);
118     }
119     printf("DISTANCE\n");
120     for(int i = 0; i < final_points.size(); i++) {
121         printf("%f\n", final_points[i].distance);
122     }
123     printf("CURVATURE\n");
124     for(int i = 0; i < final_points.size(); i++) {
125         printf("%f\n", final_points[i].curve);
126     }
127     printf("VELOCITIES\n");
128     for(int i = 0; i < final_points.size(); i++) {
129         printf("%f\n", final_points[i].vel);
130     }
131 }
132 }
133
134 std::vector<point> PurePursuitPathGen::get_points() {
135     return final_points;
136 }

```