

RIT VEXU Software Engineering Notebook

2023-2024

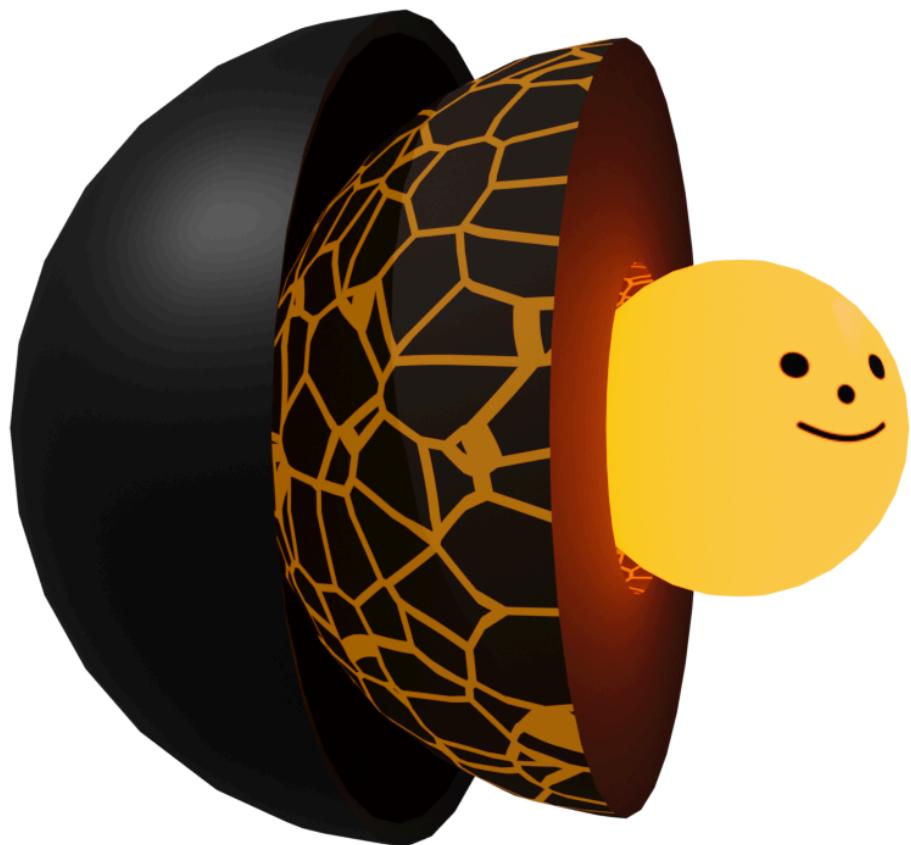


Table of Contents

RIT VEXU Software Engineering Notebook.....	1
Table of Contents.....	2
Overview.....	3
Core API.....	3
Open Source Software.....	3
Project Structure.....	4
Git Subrepo.....	4
Github Project Board.....	4
Github Actions.....	5
Auto-Notebook.....	5
Clang-Tidy.....	6
Wiki.....	6
Core: Fundamentals.....	7
Odometry.....	7
Drivetrain.....	9
Tank Drivetrain.....	9
Control Loops.....	14
Auto Command Structure (ACS).....	16
Serializer.....	18
Screen Subsystem.....	19
Catapult System.....	22
Core: Ongoing Projects.....	23
Rust.....	23
N-Pod Odometry.....	24
V5 Debug Board.....	30

Overview

Core API

All of the robot code we use is built on top of our own custom library, called the Core API, which itself is built on top of the official VEX V5 library. This API contains template code for common subsystems such as drivetrains, lifts, flywheels, and odometry, and common utilities such as vector math and command-based autonomous functions. This code remains persistent between years and is constantly updated and improved. The library can be found at github.com/RIT-VEX-U/Core

The Core codebase is abstracted in a way that allows for simple use during a hectic build season, and creates a solid foundation for future expansion. Subsystems are divided into layers following an object-oriented approach to software development.



Figure 1 - Example of Object-Oriented Programming

Open Source Software

The Core API is under the MIT open-source license, and is open for other teams to use and improve upon via pull requests. This system was modeled after the Okapi library from the Pros ecosystem, and offers similar functionality for the VexCode ecosystem. Teams that use this API are also encouraged to open source their software.

Project Structure

During the season, there are three repositories (repos) that are actively developed. Two repositories for the two competition robots, and one for the Core API. Development and code building occurs in the robot repos, and any changes to shared code (drivetrain, math utilities, major subsystems) are merged with the Core repo. This method reduces redundant code and development time.



Figure 2 - Project Structure

Git Subrepo

The Core API uses a unique type of version control called Git Subrepo (github.com/ingydotnet/git-subrepo). This allows users to simply clone the repository into an existing VexCode project to have instant access to all the tools. It also allows users to instantly receive updates by pulling from the main branch, and makes sharing code between two robot projects easier with git code merges.

Before choosing Subrepo, the team experimented with using Git Submodules to incorporate the Core API into projects. This however made Core development cumbersome and difficult for anyone unfamiliar with Git submodules specifically. Subrepo made inter-project merges more streamlined, and simplified development.

Github Project Board

In order for our software team to collaborate together with these projects, we use the Github Projects kanban-style project board. This allows us to create and assign tasks, link it to a repository and additionally notify the assigned programmer through a slackbot.

Over Under Development	In Progress	Done
<p>● Todo 20 This item hasn't been started</p> <ul style="list-style-type: none">Core #32 New Project Streamlining WikiCore #33 Core Cleanup / DocumentationCore #36 Wiki Entries	<p>● In Progress 4 This is actively being worked on</p> <ul style="list-style-type: none">Core #5 Pure Pursuit functionalityCore #34 Motion ProfilesCore #73 Accelerometer for Lateral Odometry TrackingCore #39 Add 3 Pod Odometry to Core	<p>● Done 13 This has been completed</p> <ul style="list-style-type: none">Core #44 ACS Command TimeoutCore #39 Add 3 Pod Odometry to CoreCore #43 Add Pose2D Class

Figure 3 - Software Project Board

Github Actions

This year, our team enhanced our workflow by integrating GitHub Actions into our software development process. One notable addition was an action to build our C/C++ code in the appropriate Vex environment. This automated process involves a series of steps, including checking out the repository, downloading and unzipping the Vex Robotics SDK and toolchain, and compiling the code using a Makefile. A key feature of this GitHub Action is its ability to send a Slack notification to our team channel whenever a build fails, ensuring prompt awareness and response. Furthermore, it helps maintain code integrity by preventing the merging of pull requests with failing builds. This complements our other GitHub Action for building Doxygen documentation and deploying it to GitHub Pages, allowing for seamless documentation and code management. This systematic approach aligns with our commitment to maintaining a neat, organized, and efficient engineering process.



Figure 4: Continuous Integration directly improves the quality of our code.



Figure 5: Automatically generated documentation.

Auto-Notebook

Alongside the automatic documentation, whenever Core is updated or we manually trigger it, a Github Action copies the reference manual, exports the most up to date version of our written notebook document, stitches them together, and deploys to a webpage. This is publicly available for any person wishing to see our software development process. The most valuable effect, though, is automating most of the formatting work for our notebook, work that used to require a team member to use valuable pre-competition time to sit down, append, format and export the notebook.

Clang-Tidy

In an effort to improve the quality, reduce headaches, and make our code easier to read, write, and understand, we enabled many more warnings than what is supplied with the default Vex project Makefile. These warnings deal with uninitialized variables, missing returns, and other simple code errors that nonetheless have the tendency to introduce tiny, hard to track down bugs. However, sometimes these warnings do not explore deep enough and another tool must be used. We integrated clang-tidy, a c++ linter developed by the clang compiler project, to inspect our code. With a simple switch of a variable in the Makefile, we run clang-tidy during builds which gives many insights into the code that plain compiler warnings do not. Though it does increase compilation times, it tells us about code that is bug prone or poor for performance and tests many other checks developed and validated by the wider C++ community.

Wiki

Whenever a new feature is added to Core, we create a Wiki page on the Core Github repository that provides documentation on what the function does, how to use it, and some examples of how it can be used. This documentation is easily accessible as it can be found online within the Core repository itself. This allows for new members to get acquainted with Core faster and easier than before. This allows us to speed up our training process and allow new members to start developing sooner rather than later. In addition it provides us and anyone using Core great documentation that not only goes into method signatures and descriptions, but also detailed explanations of what different methods, classes or functions do.

Opcontrol

This class provides two ways of driving the robot with a controller: Tank drive and Arcade drive. Drivers can choose what they're most comfortable with.

Tank Drive - The left joystick controls the left-side motors, and the right joystick controls the right-side motors

Arcade Drive - Acts somewhat to how modern racing video games are controlled. The left joystick controls the forward / backward speed, and the right joystick controls turning left / right.

Both functions also have an optional parameter called `power`, and refers to how the joystick is scaled to the motors. The higher the power is, the more control you have over low-speed maneuvers. Because the scaling is non-linear, it may feel weird to those who aren't used to it.

Method Signatures

```
void drive_tank(double left, double right, int power=1);
void drive_arcade(double forward_back, double left_right, int power=1);
```

Usage Examples

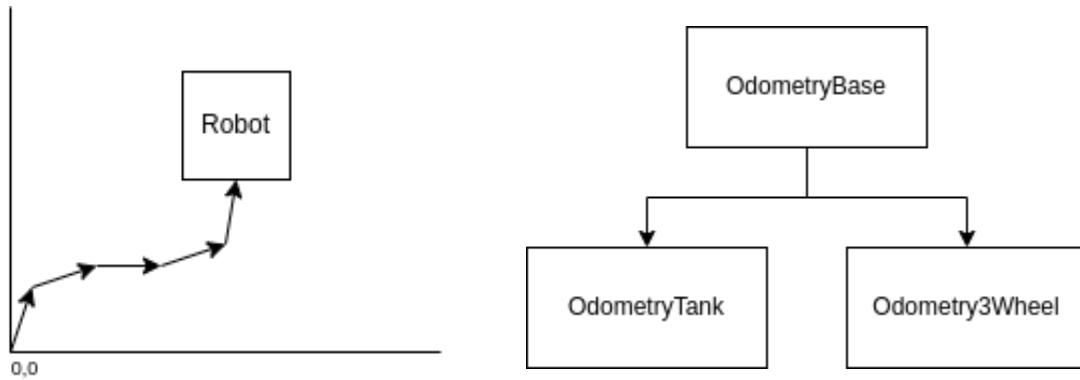
```
drive_system.drive_tank(controller.Axis3.position() / 100.0, controller.Axis2.position() / 100.0);
drive_system.drive_arcade(controller.Axis3.position() / 100.0, controller.Axis1.position() / 100.0);
```

Figure 6: A screenshot of the Core Wiki

Core: Fundamentals

Odometry

In order for the robot to drive autonomously, it needs to know where it is, and constantly monitor changes to sensors. The Odometry subsystem takes inputs from encoders, and using vector math and previous position data, calculates the position and rotation of the robot on the field as a point in space (X, Y), and heading (deg).



The Odometry subsystem is broken down into an OdometryBase class, which controls the asynchronous behavior and getters/setters, and OdometryTank and Odometry3Wheel classes, which both extend OdometryBase and implement a two-encoder algorithm and a three-encoder algorithm, respectively.

GPS + Odometry

In order to fit an 8-motor drivetrain into the 15" size requirement, the robots could not fit non-powered odometry wheels, leaving only the drive encoders to be used for position tracking. This isn't ideal, since sudden changes in acceleration and wheel slippage can easily cause the tracking to drift a substantial amount. To combat drifting, we looked to the GPS sensor for localization.

The GPS sensor uses a tag-based approach for localization, using a coded strip around the perimeter of the field to estimate position. Between pose estimates, the integrated IMU provides inertial information to estimate changes in position and heading for a constant flow of data, presumably using some sort of onboard Kalman filter. The pose (X, Y, Heading) data is sent back to the Brain over the smart port. In addition, the GPS provides a "quality" value, which is a percentage that increases when the camera can see a large amount of tape, and decreases when the camera is blocked and the IMU detects change in position over a period of time.

To properly characterize the GPS sensor, X/Y/Heading data points were gathered at different positions around the field, facing different headings. The following graphs show the data points on the 12' x 12' field grid. Distance error (in inches) to the actual measurements is shown by color.

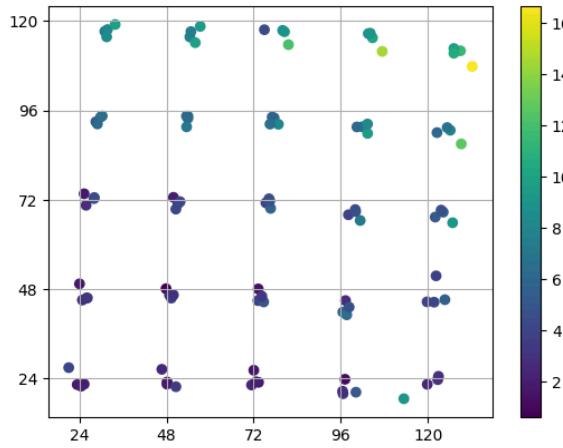


Figure 7 - Raw Data Points

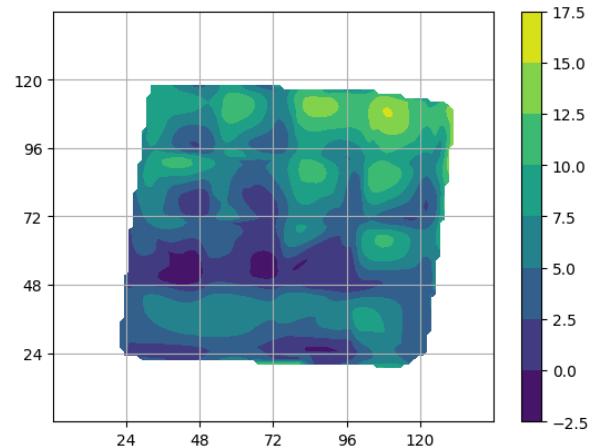


Figure 8 - Error Heat Map

There were some other errors with the GPS sensor, including issues localizing the robot when the sensor could not see enough of the coded tape. To take full advantage of the GPS sensor's localization capability, we'd need a way to perform sensor fusion alongside traditional ground-based odometry. To do this, a complementary filter was chosen - a filter that mixes two sets of data based on a proportional scalar *alpha* (α). The equation for a complementary filter is shown below:

$$out = \alpha * s_1 + (1 - \alpha) * s_2$$

where s_1 is *sensor 1*, s_2 is *sensor 2*, and *alpha* scales between the two.

To calculate *alpha*, first the X,Y position of the sensor and heading is taken into account. Since the robot will generally have a more accurate position when it's close to the wall and facing away from it, the following formula will report a score between 0 and 1 for the filter:

$$\alpha = \left(\frac{\vec{c} - \vec{p}}{\|\vec{c} - \vec{p}\|} \cdot \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix} \right) * \frac{\|\vec{c} - \vec{p}\|}{\|\vec{c}\|} * q$$

where \vec{c} is the constant vector of a point in the center of the field ($x=72$ in, $y=72$ in), \vec{p} is the robot's position as a vector (x, y), θ is the robot's heading, and x, y is again the robot's position (0 to 144 inches). The left side is the dot product of the normalized vector pointing from the robot to the center with the direction the robot is facing (gives 1 when the directions are aligned and -1 when opposite smoothly changing in between), and the right side is the sensor's distance to the center as a scalar percentage of the distance from corner to corner (between 0 and 1). Finally, q is the GPS's reported quality. We then remap this from the range [-1, 1] to [0, 1] when we do our mixing.

Drivetrain

A drivetrain class has two functions: To control the robot remotely, and autonomously. In the Core API, the TankDrive class allows the operator to control the robot using Tank controls (Left stick controls the left drive wheels, right controls the right), and Arcade controls (Left stick is forward / backwards, right stick is turning). This means drivers can tailor their controls to whichever feels more natural.

Tank Drivetrain

Brake Mode

A VEX driver has many things to keep track of during a match. From game element position, match load status, and partner robot condition there is a great deal going on. Defense is another layer on top of the mental load of playing the game. To ease this burden, we implemented a brake mode on our drive train. It is a multi-modal system that can either bring the robot to a stop or hold the robot in a specific location on the field. We use the motion profiles we developed for auto programming to decelerate the robot when requested and use our auto driving functions to hold the robot's position. We implement a smarter form of position holding than just motor braking as we can return to the exact location on the field. Additionally, we combine our deceleration control with position holding such that we do not immediately "lock the brakes" and skid away thus losing the position we attempt to hold and making driving incredibly difficult.

Autonomous Driving

For autonomous driving, the TankDrive class has multiple functions:

- `drive_forward()`:
 - Drive X inches forward/back from the current position
 - Signature: `drive_forward(double inches, directionType dir, double max_speed=1)`
- `turn_degrees()`:
 - Drive X degrees CW/CCW from the current rotation
 - Signature: `turn_degrees(double degrees, double max_speed=1)`
- `drive_to_point()`:
 - Drive to an absolute point on the field, using odometry
 - Signature: `drive_to_point(double x, double y, vex::directionType dir, double max_speed=1);`
- `turn_to_heading()`:
 - Turn to an absolute heading relative to the field, using odometry
 - Signature: `turn_to_heading(double heading_deg, double max_speed=1)`

Generally, it is better to use `drive_to_point` and `turn_to_heading` to avoid compounding errors in position over relative movements. These functions implement the `FeedbackBase` class, so any control loop can be used to control it.

Drive To Point

The defining feature of a drive to point function is the ability for a robot to calculate a relative direction and distance between its own position and the target position, and navigate to it using tuned control loops. The steps taken for our implementation are listed below.

1 - Gather information

To drive towards a specific point, the robot must know the change in angle between the robot's heading and the target, and the distance to the target. To get this, we first grab the robot's current position and heading and create a positional difference vector between this and the new point.

```
pose_t current_pos = odometry->get_position();
pose_t end_pos = { .x = x, .y = y };

point_t pos_diff_pt =
{
    .x = x - current_pos.x,
    .y = y - current_pos.y
};

Vector2D point_vec(pos_diff_pt);
```

Using this information, grab the distance to the target (using a function in the Odometry subsystem). An issue with the pure distance between points is that it does not represent how far the robot has to travel to be considered "on target" in the control loop. In order to properly reach its target, the robot should report its "aligned distance", and ignore the lateral error, as per Figure 9. This should only hold true when the robot is close to the target, or inside a given radius that is tuned by the user.



Figure 9 - Distance Modification



Figure 10 - Correction Cutoff Circle

```

double dist_left = OdometryBase::pos_diff(current_pos, end_pos);

if (fabs(dist_left) < config.drive_correction_cutoff)
{
    dist_left *= fabs(cos(angle * PI / 180.0));
}

```

The next data needed is the difference in angle between the robot's current heading and the vector between the robot's position and the target. This is calculated by using the arctangent of the difference vector, and subtracting it from the robot's current heading. The angle is then wrapped around 360 degrees.

```

double angle_to_point = atan2(y - current_pos.y, x - current_pos.x)
                           * 180.0 / PI;
double angle = fmod(current_pos.rot - angle_to_point, 360.0);
if (angle > 360)
    angle -= 360;
if (angle < 0)
    angle += 360;

double heading = rad2deg(point_vec.get_dir());
double delta_heading = 0;
if (dir == directionType::fwd)
    delta_heading = OdometryBase::smallest_angle(current_pos.rot, heading);
else
    delta_heading = OdometryBase::smallest_angle(current_pos.rot
                                                - 180, heading);

```

The last piece of information needed is whether the robot should be moving forwards or backwards. Since the distance is calculated as $\sqrt{x^2 + y^2}$, the sign is lost when squaring. Re-implement the sign based on the angle and initial driving direction.

```

int sign = 1;
if (dir == directionType::fwd && angle > 90 && angle < 270)
    sign = -1;
else if (dir == directionType::rev && (angle < 90 || angle > 270))
    sign = -1;

```

2 - Setting Control Loops

In this section, the robot takes the above information and sets its feedback loops. Since the function takes in a FeedbackBase abstract class, any feedback can be used to drive the robot's correction and linear movements. The most common situation is a trapezoidal motion profile for linear distance with PD for heading correction. Once the robot is close enough to the target point, the correction feedback is ignored to avoid issues with last-minute heading changes.

```
correction_pid.update(delta_heading);
feedback.update(sign * -1 * dist_left);

double correction = 0;
if (is_pure_pursuit || fabs(dist_left) > config.drive_correction_cutoff)
{
    correction = correction_pid.get();
}

double drive_pid_rval;
if (dir == directionType::rev) {
    drive_pid_rval = feedback.get() * -1;
} else {
    drive_pid_rval = feedback.get();
}

double lside = drive_pid_rval + correction;
double rside = drive_pid_rval - correction;

lside = clamp(lside, -max_speed, max_speed);
rside = clamp(rside, -max_speed, max_speed);

drive_tank(lside, rside);
```

Finally, when the linear feedback reports its on target, stop, return and report that the movement is over.

```
if (feedback.is_on_target())
{
    if (end_speed == 0) {
        stop();
    }
    func_initialized = false;
    return true;
}
```

Pure Pursuit

Pure Pursuit is a method of autonomous robot driving that allows the robot to autonomously drive through a set of waypoints without stopping and turning.

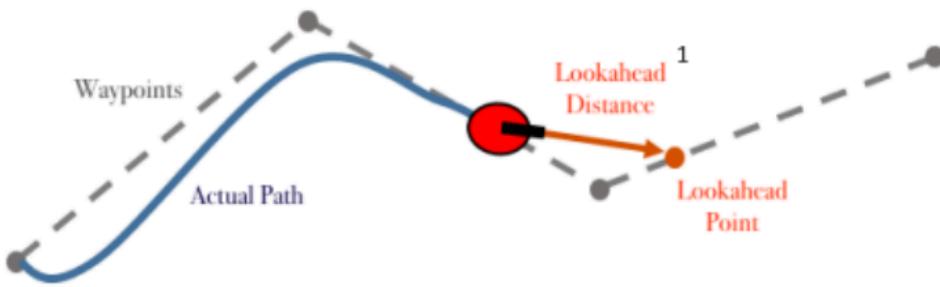


Figure 11 - Pure Pursuit Example

This is accomplished by taking a list of points (x,y), connecting them, and then choosing a "lookahead point" along the lines that the robot will attempt to follow. This will inherently cause the robot to smooth out the point map, and follow without sudden changes in direction.

The lookahead point is chosen by iterating along the path created by connecting the points and finding the furthest point that intersects a circle centered on the robot, with a set radius tuned by the programmer. Increasing this radius smooths the path, while decreasing it ensures the robot more closely follows the path.

The pure pursuit implementation in Core can either use the Autonomous Command System (ACS), or be called directly through the TankDrive class. See the sample code below for examples.

```
// Autonomous Command Controller
CommandController cmd{
    drive_sys.PurePursuitCmd(PurePursuit::Path({
        {.x=19, .y=133},
        {.x=40, .y=136},
        {.x=92, .y=136},
    }, 8), directionType::rev, .5)->withTimeout(4),
};

cmd.run();

// Standalone
while(!drive_sys.pure_pursuit(PurePursuit::Path({
    {.x=19, .y=133},
    {.x=40, .y=136},
    {.x=92, .y=136}}, 8), directionType::fwd, 0.5))
{
    vexDelay(20);
}
```

Control Loops

In order for the Autonomous Command Structure to function, we need a way to tell the robot how we want it to move. There are two broad categories of telling a robot to achieve a requested position - Feedback and Feedforward. Feedback relies on sensors and adjusts the output of the robot according to the error between where it is and where it wants to be. On the other hand, a feedforward controller takes a mathematical model of the system and creates outputs based on what it calculates to be the necessary output to achieve the goal. Additionally, there are simpler methods like Bang-Bang or Take Back Half. These adjust the outputs based on the current position relative to the target, where Take Back Half gradually refines the output until it settles at the desired position. These controller types work for many applications, but a combination of them can achieve an even better control over robot actuators.

PID

A PID controller is perhaps the most common type of Feedback control. It uses measurements of the error at its current state (proportional), measurements of how the error was in the past (integral) and measurements of how the error changes over time(derivative). The controller acts accordingly to bring the errors towards 0. We implemented a standard PID controller but made some alterations to fit our needs. The most important of these are custom error calculations. The standard error calculation function (*target - measured*) works for many of our uses but causes problems when we use a PID controller to control angles. Since angles wrap around at 360 degrees or 2π radians we wrote our own error calculation function that gives the error that accounts for this wrapping.

Feedforward

A feedforward controller differs from a feedback controller in that it does not rely on any measurement of error to command a system. Instead, built into a feedforward controller is a mathematical model of the domain. When a target is requested by the controller, the model is queried to figure out what the robot actuators must output to achieve that target. A key advantage of this form of control is that instead of waiting for an error to build up in the system, the controller acts directly to achieve the target and can reach the target much faster.

Bang-Bang

Bang-Bang control is a straightforward control methodology where the output to the system is either fully on or fully off, with no intermediate states. It's used for systems where fine control isn't necessary or possible. In this method, when the process variable is below the setpoint, the controller output is set to maximum; when above, it's set to minimum. This approach is simple and often used for systems with high inertia or where the precise control of the variable isn't critical. However, it can lead to oscillations around the setpoint and isn't suited for systems requiring precise regulation.

Take Back Half (TBH)

The Take Back Half (TBH) method is an iterative approach used to refine control in systems where overshoot is a concern. This method adjusts the output by taking back half the value of the output each time the controlled variable overshoots the target. The adjustment continues until the system settles close to the desired setpoint. TBH is particularly useful in scenarios where a fine balance between responsiveness and stability is needed, as it reduces the oscillation or overshoot often seen in simpler control methods. It's a practical choice for systems where a PID controller might be too complex or unnecessary. TBH controllers only have one tuning parameter which allows for an incredibly easy tuning experience.

Generic Feedback

Different control systems work best in different environments. Because of this, we found ourselves switching control schemes often enough that rewriting the code each time was time consuming and often led to rushed, worse quality code. To solve this problem we implemented a generic feedback interface so that none of our subsystem code needs to change when we use a different control scheme. Instead, the subsystem reports to the controller where it wants to be, measurements from its environment and some information about the system's capabilities and the controller will report back the actions needed to achieve that target. This allows for much faster prototyping as well as cleaner, less tightly coupled code.

Motion Profile

As we learn from each event, our team has evolved our approach to robot control systems, transitioning from a simple PID controller to a more sophisticated Motion Profile controller. The PID system, while fundamental, had its drawbacks, such as limited speed specification, poor response to wheel slipping, and slower reaction times. These limitations highlighted the need for a more advanced control mechanism.

Our Motion Profile controller represents a significant upgrade. It integrates precise control over position, acceleration, and velocity, allowing for optimized performance of our robot's subsystems. Unlike the PID controller, which reacts only to discrepancies between actual and desired states, our Motion Profile controller proactively manages the robot's movements. It anticipates the required actions, thereby reducing response lags. Moreover, it avoids the rigidness of a pure feedforward controller by adapting dynamically to changing conditions in competition scenarios.



Figure 12: Trapezoidal motion profile

A key feature of the Motion Profile controller is its ability to handle varying accelerations. This functionality enables our robot to accelerate efficiently without wheel slipping, always maintaining optimal acceleration. This year, we've further refined our Motion Profile to accommodate non-zero starting and ending velocities. This enhancement allows for the seamless chaining of complex movements, ensuring smoother transitions and more fluid motion during competition tasks.

Auto Command Structure (ACS)

Principle

A recent addition to our core API was that of the Autonomous Command Structure. No more will our eyes glaze over staring at brackets as we trawl through an ocean of anonymous functions nor lose our way in a labyrinthine state machine constructed not of brick and stone but blocks of ifs and whiles. Instead, we provide named Commands for all the actions that our robot can execute and infrastructure to run them sequentially or concurrently. The API is written in a declarative way allowing even programmers unfamiliar with the code to see a step-by-step, annotated guide to our autonomous path while keeping the procedures of how to execute the actions from hurting the readability of the path.

```

CommandController auto_non_loader_side(){
    int non_loader_side_full_court_shot_rpm = 3000;
    CommandController non_loader_side_auto;

    non_loader_side_auto.add(new SpinRPMCommand(flywheel_sys, non_loader_side_full_court_shot_rpm));
    non_loader_side_auto.add(new WaitUntilUpToSpeedCommand(flywheel_sys, 10));
    non_loader_side_auto.add(new ShootCommand(intake, 2));
    non_loader_side_auto.add(new FlywheelStopCommand(flywheel_sys));
    non_loader_side_auto.add(new TurnDegreesCommand(drive_sys, turn_fast_mprofile, -60, 1));
    non_loader_side_auto.add(new DriveForwardCommand(drive_sys, drive_fast_mprofile, 20, fwd, 1));
    non_loader_side_auto.add(new TurnDegreesCommand(drive_sys, turn_fast_mprofile, -90, 1));
    non_loader_side_auto.add(new DriveForwardCommand(drive_sys, drive_fast_mprofile, 2, fwd, 1));
    non_loader_side_auto.add(new SpinRollerCommand(roller));

    return non_loader_side_auto;
}

```

Figure 13: ACS code from the 2023 competition season

Updates

This season, we found ourselves annoyed with having to repeat basic things such as `path.add(...)` and having to write `new ThingCommand(...)` over and over again. Our first solution to this was “shortcuts”. These were member functions of subsystems that would allocate, initialize and return an auto command for that subsystem. So, instead of `path.add(new DriveForwardCommand(drive_sys, drive_fast_mprofile, 20, fwd))` we could simply write `path.add(drive_sys.DriveForwardCommand(20, fwd))`. This reduced a great deal of typing but still left us with some issues.

The most hazardous, rather than the simply annoying downside of last year's system, was the memory unsafety of this system. Since our auto commands must use virtual functions, they must be on the other end of a pointer. So, they must be allocated using `new` or they must be initialized statically before we write the path which is a terrible user experience (Though, if constrained by an embedded system where allocating on the heap was deemed dangerous, the system could work with this). This became a real issue when we began to write more complicated constructs such as branching, asynchronous, and repeated commands as it became dangerously unclear who was responsible for deallocating these objects. As a solution for this, we developed an RAI wrapper for the Auto Command Interface. Inspired by C++'s `std::unique_ptr`, this wrapper provides a memory safe, value based way of using auto commands while still maintaining their adaptability. We used C++'s ideas of move semantics and ‘Resource Allocation Is Initialization’ to practically solve memory management so programmers (and even non programmers) can focus on writing paths.

```

CommandController cmd{
    odom.SetPositionCmd({.x = 16.0, .y = 16.0, .rot = 225}),
    // 1 - Turn and shoot preload
    {
        cata_sys.Fire(),
        drive_sys.DriveForwardCmd(dist, REV),
        DelayCommand(300),
        cata_sys.StopFiring(),
        cata_sys.IntakeFully(),
    },
    // 2 - Turn to matchload zone & begin matchloading
    drive_sys.DriveForwardCmd(dist + 2, FWD, 0.5)
        .with_timeout(1.5),

    // Matchloading phase
    Repeat{
        odom.SetPositionCmd({.x = 16.0, .y = 16.0, .rot = 225}),
        intakeToCata.with_timeout(1.75),
        cata_sys.Fire(),
        drive_sys.DriveForwardCmd(10, REV, 0.5),
        cata_sys.StopFiring(),
        cata_sys.IntakeFully(),
        drive_sys.TurnToHeadingCmd(load_angle, 0.5),
        drive_sys.DriveForwardCmd(12, FWD, 0.2).with_timeout(1.7),
    }.until(TimeSinceStartExceeds(30))
};

}

```

Figure 13: ACS code going into the 2024 competition season

Now that we were free to use auto commands without fear for leaking memory or messing with currently running commands, we began to create more powerful constructs such as branching on runtime information, timeouts so the robot can decide what to do based on how much time is left in the auto or skills period, fearless concurrency (driving and reloading at the same time), and a much much nicer user interface. This declarative, safe, and straightforward method of writing auto paths lets us spend less time writing and debugging custom code and more time exploring and optimizing auto paths.

Serializer

One pain point we found last year was configuring auto paths, color targets, path timeouts, and other parameters that changed often but for the most part should be persistent. Commonly, we found ourselves redeploying code at the last minute before a match. To solve this, we wrote a class that takes control of a file on the SD card to which users can read and write values at runtime using a simple key-value interface. This keeps us from having to change a value, redeploy, repeat which cost us valuable time in the past.

Screen Subsystem

Principle

One of the most powerful elements of the V5 Brain is the fairly substantial touch screen. However, its simple drawing API limits its utility as one person's part of the code will draw over another since there is no larger abstraction controlling who draws when. We have many different subsystems on our robot to observe and debug and many parameters that can be tuned at run time and the screen provides a way to do this. We provide an API that provides a 'page' interface that can be inserted into a slideshow-like interface. Each 'page' provides two functions, an update and a draw. The update runs more frequently allowing touch input and data collection at a reasonably fast rate while the draw function runs less frequently to not cause too much overhead on the system. At startup, users provide the screen subsystem a list of pages and the screen subsystem handles orchestration and input in a background thread while other robot code runs unaffected.

```
pages = {
    new AutoChooser({"Auto 1", "Auto 2", "Auto 3", "Auto 4"}),
    new screen::StatsPage(motor_names),
    new screen::OdometryPage(odom, 12, 12, true),
    cata_sys.Page(),
};

screen::start_screen(Brain.Screen, pages);
```

Figure 14: Configuration for the screen subsystem

Pages

Odometry Page

The odometry page has proved incredibly useful in writing and debugging auto and auto skills paths. It shows a picture of the robot on the field as well as a print out of the actual x,y coordinates and heading of the robot. Since we write our autos with respect to the coordinate system of the field, having a map to look at makes development much simpler.



Figure 15: A field display for the Over Under season

PID Tuner

PID controllers are integral to many subsystems on our robots. Our drive code uses them for turning and forward motion, our catapult uses them for reloading, and subsystems across seasons require them for precise control. Tuning them, however, can be incredibly tedious. Changing one value, redeploying, and repeating over and over again is time consuming and unnecessary. Since we have a wonderful touchscreen, we simply added a series of sliders for PID parameters and we can now easily adjust a PID tuning in seconds rather than minutes saving a great deal of time on an already time consuming part of robot development.

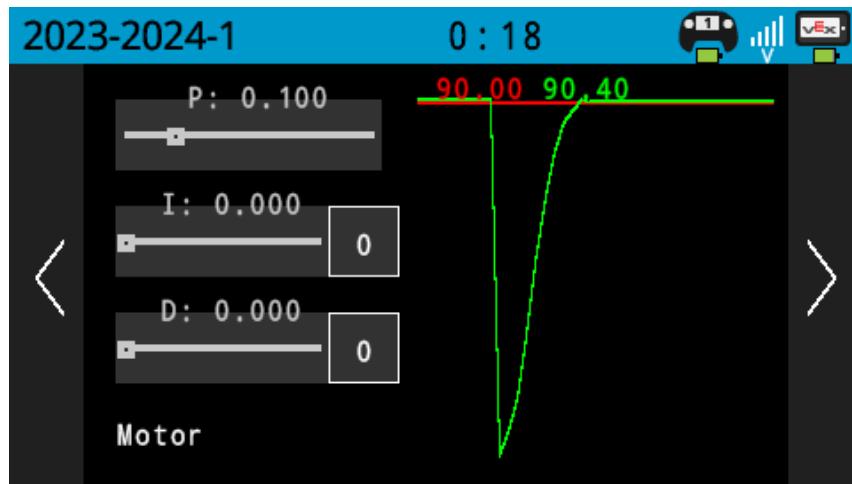


Figure 16: Tuning a motor for reaching an angle

Motor Stats

One would think it an easy step to remember to plug motors in, and yet multiple times this season we have been bewildered and hindered by an unplugged motor. This page was written to continuously display that the motor had been unplugged and was not cancellable like the built-in VEX alert. This screen also displays what port to plug it into as well as a color coded

temperature displaying when the robot needs to cool down. This tool proved extremely useful as we discovered an alarmingly high number of dead or nonfunctioning ports on the brain.



Figure 17: Motor Stats screen from our 2023 robot

Cata System Page



Figure 18: The catapult status page. Includes a graph of Catapult PID values.

Catapult System

Vex's Over Under game requires the effective utilization of the fascinatingly shaped triball. After much deliberation, our team decided on a catapult to launch the game element across the field and a reversible intake for picking up and scoring the triball. This system gives us a great deal of flexibility and power for strategy but does increase the system's complexity. This complexity mostly stems from the orchestration of intaking with catapult reloading such that we do not jam our catapult and never intake multiple game pieces leading to a disqualification.

We implemented a state machine that receives inputs from the controller, a distance sensor in the intake, a distance sensor in the catapult, and a potentiometer for watching the catapult's position. The system runs the appropriate motors to either intake to hold the triball, reset the catapult, intake into the catapult, or shoot depending on its state. Because we have so many sensors, we can determine when intaking would lead to disqualification and simply not honor the intaking command.

These messages are a simple enum that one passes to `CataSys::send_command()`. This was originally intended to make writing multi-threaded code less error-prone as there was one thread-safe and simple way to interact with the subsystem, rather than many disparate methods some of which are meant for internal usage of the class on the running thread and some accessors and setters meant to be used from the user thread. Although it started for implementation ease, it naturally brought about a very simple interface for auto. Instead of sending a command on a button press, we simply send a command at a certain point in our auto path and the system reacts accordingly.

Core: Ongoing Projects

Rust

Over the course of the year, we have experimented with rewriting our Core API in Rust, a multi-paradigm programming language focused on performance and safety. Rust offers several potential advantages over C++:

Memory Safety

One of the primary benefits of Rust is its emphasis on memory safety without sacrificing performance. Rust's ownership model ensures that memory is managed correctly at compile time, reducing the risk of memory leaks and buffer overflows which are common issues in C++. This is especially crucial in robotics, where memory management errors can lead to system crashes or unpredictable behavior in real-time operations.

Concurrency

Rust's approach to concurrency is another major advantage. Concurrency errors, like race conditions, are hard to debug and can be catastrophic in robotics, leading to inconsistent states and erratic behavior. Rust's type system and ownership model prevent data races at compile time, making concurrent programming more reliable and easier to reason about.

Performance

In terms of performance, Rust is comparable to C++, which is essential in robotics where processing speed and response time are critical. Rust's zero-cost abstractions mean that high-level constructs do not add overhead at runtime. This allows developers to write high-level code without compromising on performance, an important consideration in robotics where every millisecond can count.

Improved Code Maintenance and Readability

Rust also offers improved code maintainability and readability. Its modern syntax and language features make it easier to write clear and concise code. This reduces the cognitive load on developers, making it easier to develop and maintain complex robotic systems. The compiler's strictness also ensures that many potential bugs are caught early in the development cycle, reducing the time spent on debugging.

Growing Ecosystem and Community

The Rust ecosystem is rapidly growing, with a strong focus on safety and performance. There are increasing numbers of libraries and tools being developed for Rust, including those specifically for robotics. The Rust community is known for its dedication to improving code quality and security, which aligns well with the needs of robotics development.

Overall

While the transition from C++ to Rust in a robotics context requires investment in terms of learning and codebase modification, the benefits in memory safety, concurrency handling, performance, and maintainability make it a compelling choice. The modern features of Rust, combined with its growing ecosystem and community support, position it well for developing robust, efficient, and safe robotic systems.

Progress

Though progress slowed as competition season began, our rust build system is moving out of the proof of concept stage and into something useful. We setup a cargo (rust's build system manager) target and can cargo build a vex project into an executable file. We then created a simple python script to convert the .elf file into the stripped binary file that the vex brain expects and use the vexcom tool provided by the vscode extension to

N-Pod Odometry

Motivation

Although we have been working on the GPS odometry system, wheel odometry is still vital. It provides great small-scale, quickly updating positions as well as having near-perfect, continuous, local velocity which a GPS system can not achieve. We use odometry in two ways; either tank or differential odometry where there is one wheel on either side of the drivetrain alongside the drive wheels and 3-wheel odometry where we have 3 wheels at ninety degrees to each other. Tank odometry is limited as it can not track horizontal movement and we simply hope that we never move sideways, though it is easiest to implement in the robot so it is our most commonly used system. 3-wheel odometry solves the side-to-side problem but is much harder to implement in hardware owing to the extra wheel where other subsystems would need space.

In a plea for mercy from the hardware team, we agreed that we would take tracking wheels wherever and we could make do. Though we once again got stuck with a tank system, if our dream of more tracking wheels ever comes true we would need code to handle such a system. Also, since tank and 3 wheel odometry are special cases of an n-pod system, we could reduce code duplication.

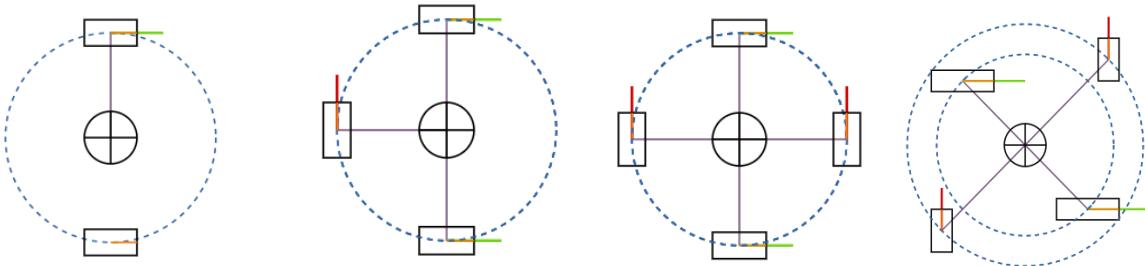


Figure 19: 2 pod, 3 pod, and arbitrary pods such a system could handle.

Syntax

After much brainstorming and many mad scientist whiteboard drawings, we believe that we have the fundamentals of a system figured out. Unfortunately, other responsibilities to the team came up so we do not yet have a functioning implementation of the system.

Imagine a robot with n number of tracking omni wheels. We could read encoder values E_1, E_2, \dots, E_n from the system in radians from the initial position. As well, each encoder has a configuration $(x_1, y_1, \theta_1, r_1), \dots, (x_n, y_n, \theta_n, r_n)$ describing its position (x, y) relative to the center of rotation, an angle describing its orientation relative to the robot frame (θ), and a radius of the wheel (r).

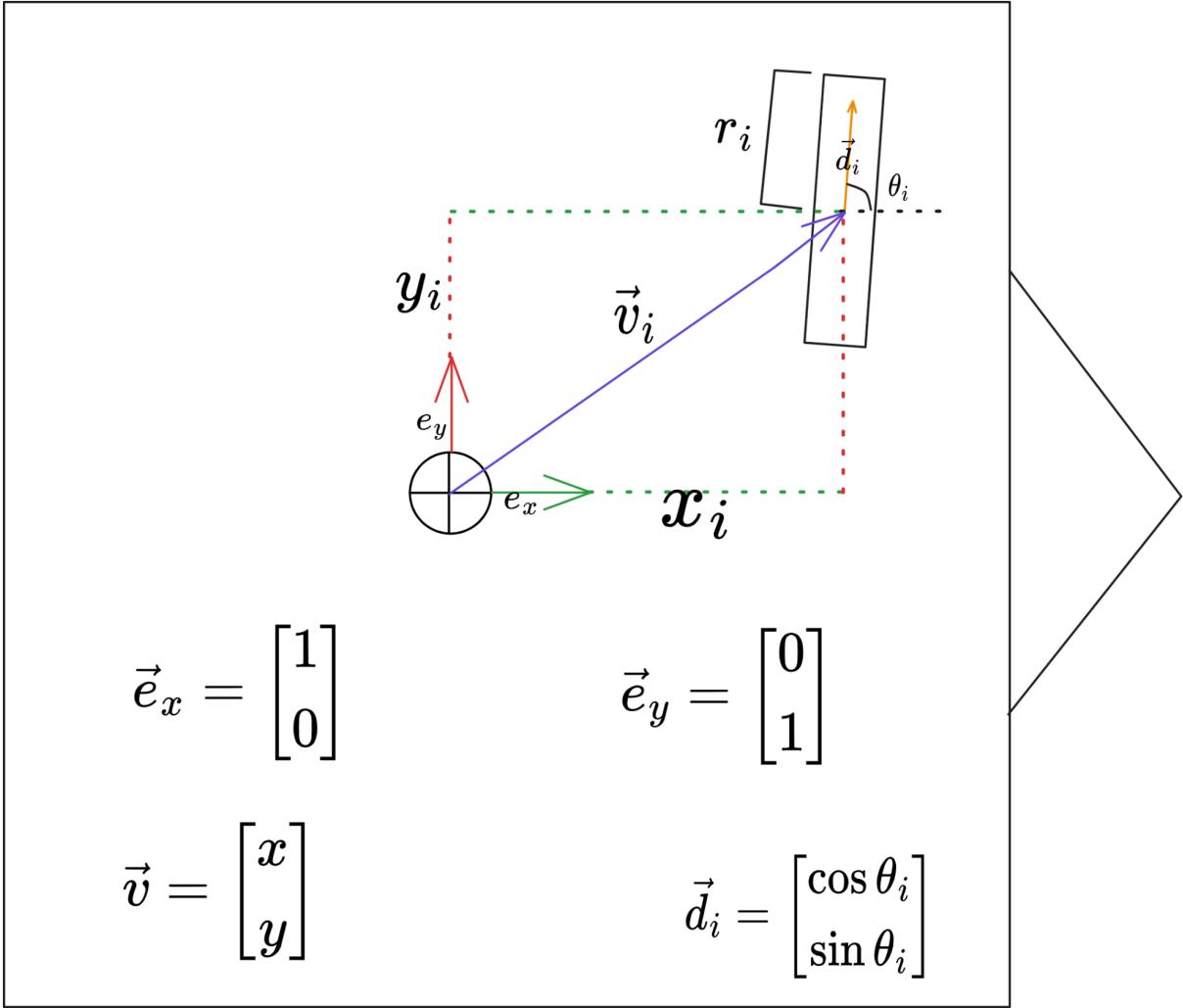


Figure 20: The configuration of a tracking wheel on the robot. (e_x, e_y) are the basis vectors of our coordinate system - the X and Y axes of the robot coordinate frame. d_i is the direction vector of the tracking wheel.

Now, if we pretend that these wheels are powered and we wish to translate and rotate the robot according to some controller input (x, y, θ) we can develop a formula for how much each wheel needs to rotate to move the robot in that direction with that rotation. Luckily, since the tracking wheels are omni wheels that roll freely in the axis against their “forward” direction, we do not need to worry about dragging a wheel so long as it is spinning the correct amount in its “forward” direction. For a desired (x, y, θ) (in the robots reference frame), for the i -th encoder, we say $E_i = xF_{xi}$. That is, for a movement in the x-axis the rotations of the i -th encoder, are the desired x movement times some scalar factor (F) for how far this specific wheel would rotate. Similarly, for a y only and θ only movement, $E_i = yF_{yi}$ and $E_i = \theta F_{\theta i}$ respectively.

Deriving Factors

F_x

F_x depends on the direction vector \vec{d} of the omni-wheel. If the omni-wheel is facing along the x-axis, F_x will be higher whereas if the omni-wheel is directly perpendicular to the x-axis, it will not spin when you move only in the x-direction. Since \vec{e}_x and \vec{d} are unit vectors, how closely they are related is given by $\vec{e}_x \cdot \vec{d} = \cos(\text{angle between } x \text{ axis and wheel})$

F_x also depends on the radius of the wheel r . One full rotation of the wheel moves a distance of $C = 2\pi r$. If we drive in the direction of the wheel i inches, the wheel will complete $\frac{i}{2\pi r}$ revolutions. If we measure the rotations in radians, the wheel will travel $\frac{i}{r}$ radians. That is, if the encoder wheel travels E radians, we will have traveled Er inches in that direction.

So, the distance traveled in the x direction of a wheel pointing in the direction \vec{d} , rotating E radians is $x = Er(\vec{e}_x \cdot \vec{d})$. This gives since F_x as how many inches per radian turned,

$$F_x = \frac{x}{E} = r(\vec{e}_x \cdot \vec{d})$$

F_y

F_y is derived almost identically as F_x just instead of testing against \vec{e}_x we test against \vec{e}_y . So,

$$F_y = \frac{y}{E} = r(\vec{e}_y \cdot \vec{d})$$

F_θ

F_θ is a little more complicated since it is determined by the position of the wheel \vec{v} as well as the orientation of the wheel \vec{v}

Imagine that the robot turns an angle of θ_r measured in radians. A wheel that is perfectly perpendicular to the rotation will travel an arc with distance $S = ||\vec{v}|| \theta_r$ by the arc length formula where the 'radius' of the arc is defined by the length of the vector \vec{v} .

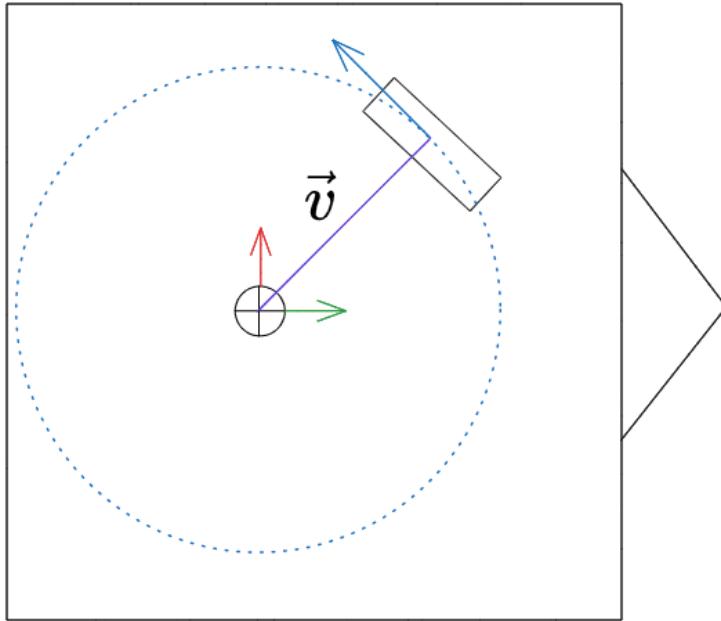
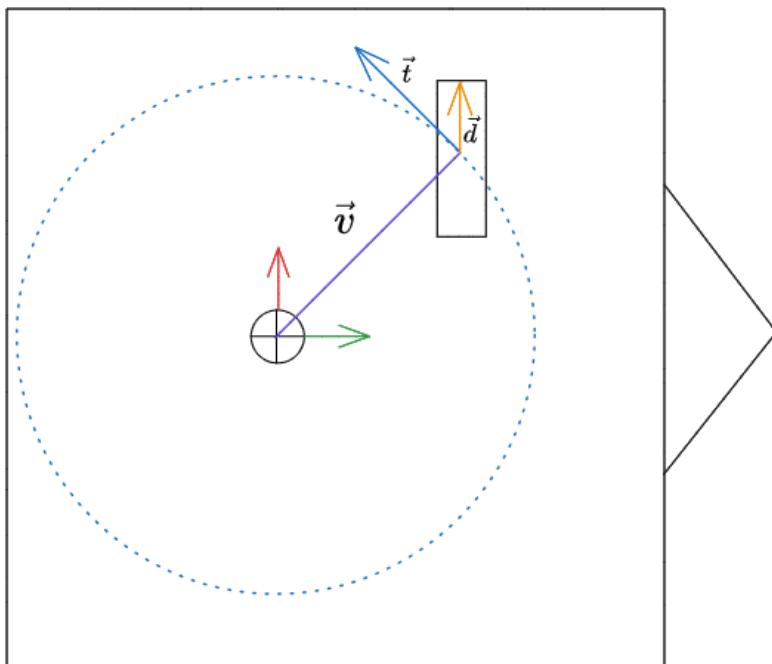


Figure 21: A conceptual perfectly perpendicular wheel

So, if we have a wheel that is always tangent to the rotation, it will travel

$$E_t r = S = \vec{v} \cdot \vec{t}$$

Since our wheel isn't guaranteed to be perfectly tangent to the arc, we have to use our dot product trick to get the component of its motion that is tangent to the turning circle. That is, instead of comparing to \vec{e}_x or \vec{e}_y we compare to the normalized vector \vec{t} tangent to the turning circle.



$$E_t r = Er(\vec{t} \cdot \vec{d})$$

So

$$Er(\vec{t} \cdot \vec{d}) = S = ||\vec{v}||\theta_t$$

Since \vec{t} is just a unit vector 90 degrees counterclockwise of \vec{v} , We can find it by multiplying \vec{v} by the rotation matrix for 90 degrees and normalizing giving

$$\vec{t} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} norm(\vec{v}) = \begin{bmatrix} -norm(\vec{v}).y \\ norm(\vec{v}).x \end{bmatrix}$$

So

$$F_\theta = \frac{\theta_r}{E} = \frac{r(\vec{t} \cdot \vec{d})}{||\vec{v}||} = \frac{r(\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} norm(\vec{v}) \cdot \vec{d})}{||\vec{v}||} = \frac{r(\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \vec{v} \cdot \vec{d})}{||\vec{v}||^2}$$

Why factors

These factors make solving this problem much simpler. For the forward case, for some wheel i , its rotation is the sum of all the motions applied to it. So $E_i = F_{xi}x + F_{yi}y + F_{\theta i}\theta$. So, for all the wheels, we plug in the commanded (x, y, θ) to each wheel's factors to get its necessary rotation. Since the factors depend only on the wheel's pose in the frame, these can be calculated once at the start of the program and are constant (unless the frame breaks apart, in which case the robot has other problems).

Now Do It Backwards

We have now solved the forward system for when we have a delta of our pose and want our wheel deltas. Now we must take our wheel deltas and solve for our pose delta. We have our formulas for each wheel's encoder motion and can consider this as a system of linear equations. At runtime, we have our wheel encoder deltas we can plug in and then we can solve the system of linear equations. This requires that we have enough data to satisfy the equations. That is, we need at least 3 separate wheels with at least some angle between them, or else the system will be not fully constrained. In the case of tank odometry, we only have two wheels but as outside observers we know we can not measure change in one dimension. So, we know one variable is zero and then have two remaining free variables and two equations to satisfy the system. For robots with greater than three encoders, we have an over-constrained system of equations but this is not an issue. Since all the encoders are modeled on a physical system, they should agree on what the solution is. Using the technique of least squares regression, we can find our (x, y, θ) to solve the over-constrained system that minimizes the error between equations. This

also gives us a way to detect errors in our drive train. If a wheel gets jammed, its encoder reading will disagree with the rest of the system, and the error value will measurably increase. If we monitor this error value we can diagnose mechanical or electrical issues from the code.

$$T = \begin{bmatrix} F_{x1} & F_{y1} & F_{\theta 1} \\ \vdots & \vdots & \vdots \\ F_{xn} & F_{yn} & F_{\theta n} \end{bmatrix}$$

$$\vec{X} = \begin{bmatrix} \frac{dx_{robot}}{dt} \\ \frac{dy_{robot}}{dt} \\ \frac{d\theta_{robot}}{dt} \end{bmatrix}$$

$$\vec{E} = \begin{bmatrix} E_1 \\ \vdots \\ E_n \end{bmatrix}$$

$$\begin{bmatrix} \text{Length} & \text{Length} & \text{Angle} \\ \text{Angle} & \text{Angle} & \text{Angle} \\ \vdots & \vdots & \vdots \end{bmatrix}$$

transfer matrix
from robot velocity
to encoder velocities
(f for factor)

$$\begin{bmatrix} \frac{\text{Distance}}{\text{Time}} \\ \frac{\text{Distance}}{\text{Time}} \\ \frac{\text{Angle}}{\text{Time}} \end{bmatrix}$$

pose velocity

$$\begin{bmatrix} \frac{\text{Angle}}{\text{Time}} \\ \vdots \\ \frac{\text{Angle}}{\text{Time}} \end{bmatrix}$$

encoder wheel
velocities

$$T\vec{X} = \vec{E}$$

$$\vec{X} = T^{-1}\vec{E}$$

or in the case where the matrix is not invertible, find the best solution

The linear algebra behind the solution

V5 Debug Board

The large number of features added to core, while extremely useful, are also very difficult to debug. Without a proper real-time c++ debugger and one stream serial data for print statements, data parsing can get very messy. The improvements to the Screen subsystem have helped, but a remote solution is needed to avoid chasing after the robot to get visual data.



Figure 22 - Debug Board (Back)

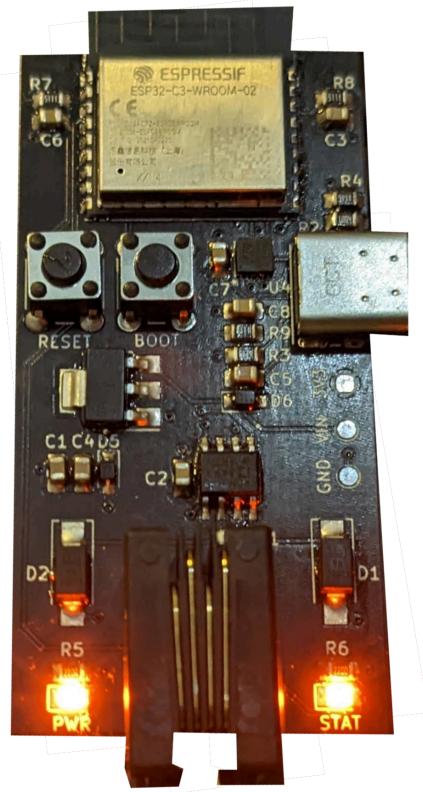


Figure 23 - Debug Board (Front)

The V5 Debug Board is a custom PCB designed by our team specifically to interface with the V5 Smart Ports, host a ROS2 node and a WiFi access point for programmers to connect to with laptops. This board is designed to ingest any kind of data and use it for graphs, real-time tuning and even displaying a 3D model of the robot on a virtual field using odometry data. This data can be viewed using either ROS' RViz or Foxglove visualization software.

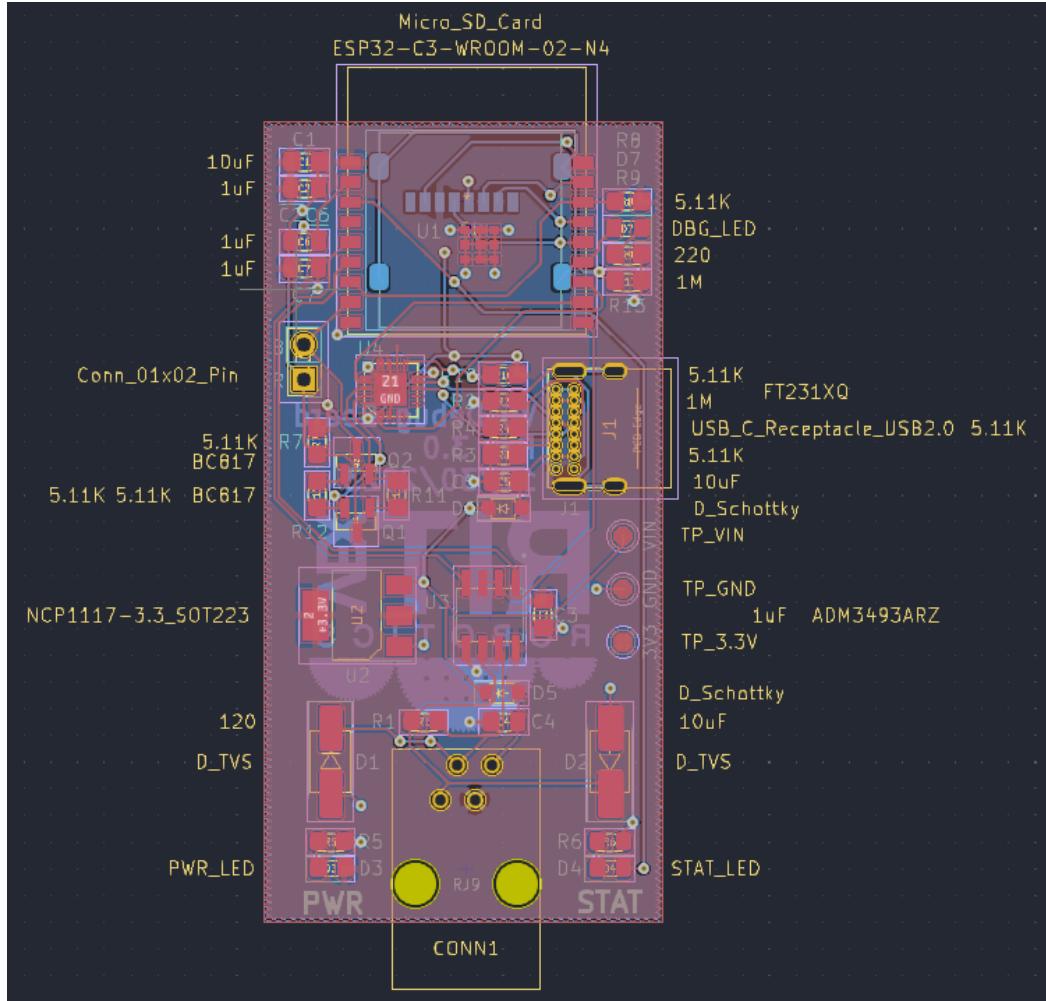


Figure 24 - Debug Board PCB Layout

Hardware design is nearing completion, with 3 revisions built and tested. Revision 3.0 is powered by an ESP32-C3-WROOM2 microcontroller, and uses an RS-485 transceiver to communicate with the Brain over a smart-port. The new addition of a Micro-SD card allows users to upload their own 3D model of the robot, and provides data logging capabilities.

As of now, the hardware design is nearly complete. Software has achieved WiFi AP broadcasting, TCP communications and work is starting on the Micro-ROS implementation. The design is fully open source under the GPL-3 license, and is hosted at github.com/superrm11/VexDebugBoard.

RIT VEXU Core API

Generated by Doxygen 1.10.0

1 Core	1
1.1 Getting Started	1
1.2 Features	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	7
3.1 Class List	7
4 File Index	11
4.1 File List	11
5 Class Documentation	13
5.1 AndCondition Class Reference	13
5.1.1 Member Function Documentation	13
5.1.1.1 test()	13
5.2 Async Class Reference	14
5.2.1 Detailed Description	14
5.2.2 Member Function Documentation	15
5.2.2.1 run()	15
5.3 AutoChooser Class Reference	15
5.3.1 Detailed Description	16
5.3.2 Constructor & Destructor Documentation	16
5.3.2.1 AutoChooser()	16
5.3.3 Member Function Documentation	16
5.3.3.1 draw()	16
5.3.3.2 get_choice()	17
5.3.3.3 update()	17
5.3.4 Member Data Documentation	17
5.3.4.1 choice	17
5.3.4.2 list	17
5.4 AutoCommand Class Reference	18
5.4.1 Member Function Documentation	19
5.4.1.1 on_timeout()	19
5.4.1.2 run()	19
5.4.2 Member Data Documentation	19
5.4.2.1 timeout_seconds	19
5.5 BangBang Class Reference	20
5.5.1 Member Function Documentation	20
5.5.1.1 get()	20
5.5.1.2 init()	20
5.5.1.3 is_on_target()	21
5.5.1.4 set_limits()	21

5.5.1.5 update()	21
5.6 BasicSolenoidSet Class Reference	22
5.6.1 Detailed Description	22
5.6.2 Constructor & Destructor Documentation	22
5.6.2.1 BasicSolenoidSet()	22
5.6.3 Member Function Documentation	23
5.6.3.1 run()	23
5.7 BasicSpinCommand Class Reference	23
5.7.1 Detailed Description	24
5.7.2 Constructor & Destructor Documentation	24
5.7.2.1 BasicSpinCommand()	24
5.7.3 Member Function Documentation	25
5.7.3.1 run()	25
5.8 BasicStopCommand Class Reference	25
5.8.1 Detailed Description	26
5.8.2 Constructor & Destructor Documentation	26
5.8.2.1 BasicStopCommand()	26
5.8.3 Member Function Documentation	26
5.8.3.1 run()	26
5.9 Branch Class Reference	27
5.9.1 Detailed Description	28
5.9.2 Member Function Documentation	28
5.9.2.1 on_timeout()	28
5.9.2.2 run()	28
5.10 screen::ButtonConfig Struct Reference	28
5.11 screen::ButtonWidget Class Reference	28
5.11.1 Detailed Description	29
5.11.2 Constructor & Destructor Documentation	29
5.11.2.1 ButtonWidget() [1/2]	29
5.11.2.2 ButtonWidget() [2/2]	29
5.11.3 Member Function Documentation	30
5.11.3.1 update()	30
5.12 screen::CheckboxConfig Struct Reference	30
5.13 CommandController Class Reference	30
5.13.1 Detailed Description	31
5.13.2 Constructor & Destructor Documentation	31
5.13.2.1 CommandController()	31
5.13.3 Member Function Documentation	31
5.13.3.1 add() [1/3]	31
5.13.3.2 add() [2/3]	32
5.13.3.3 add() [3/3]	32
5.13.3.4 add_cancel_func()	32

5.13.3.5 add_delay()	33
5.13.3.6 last_command_timed_out()	33
5.13.3.7 run()	33
5.14 Condition Class Reference	33
5.14.1 Detailed Description	34
5.15 CustomEncoder Class Reference	34
5.15.1 Detailed Description	34
5.15.2 Constructor & Destructor Documentation	34
5.15.2.1 CustomEncoder()	34
5.15.3 Member Function Documentation	35
5.15.3.1 position()	35
5.15.3.2 rotation()	35
5.15.3.3 setPosition()	35
5.15.3.4 setRotation()	36
5.15.3.5 velocity()	36
5.16 DelayCommand Class Reference	36
5.16.1 Detailed Description	37
5.16.2 Constructor & Destructor Documentation	37
5.16.2.1 DelayCommand()	37
5.16.3 Member Function Documentation	37
5.16.3.1 run()	37
5.17 DriveForwardCommand Class Reference	38
5.17.1 Detailed Description	39
5.17.2 Constructor & Destructor Documentation	39
5.17.2.1 DriveForwardCommand()	39
5.17.3 Member Function Documentation	39
5.17.3.1 on_timeout()	39
5.17.3.2 run()	40
5.18 DriveStopCommand Class Reference	40
5.18.1 Detailed Description	41
5.18.2 Constructor & Destructor Documentation	41
5.18.2.1 DriveStopCommand()	41
5.18.3 Member Function Documentation	41
5.18.3.1 on_timeout()	41
5.18.3.2 run()	41
5.19 DriveToPointCommand Class Reference	42
5.19.1 Detailed Description	43
5.19.2 Constructor & Destructor Documentation	43
5.19.2.1 DriveToPointCommand() [1/2]	43
5.19.2.2 DriveToPointCommand() [2/2]	43
5.19.3 Member Function Documentation	44
5.19.3.1 run()	44

5.20 AutoChooser::entry_t Struct Reference	44
5.20.1 Detailed Description	44
5.20.2 Member Data Documentation	44
5.20.2.1 name	44
5.21 ExponentialMovingAverage Class Reference	45
5.21.1 Detailed Description	45
5.21.2 Constructor & Destructor Documentation	45
5.21.2.1 ExponentialMovingAverage() [1/2]	45
5.21.2.2 ExponentialMovingAverage() [2/2]	46
5.21.3 Member Function Documentation	46
5.21.3.1 add_entry()	46
5.21.3.2 get_size()	46
5.21.3.3 get_value()	46
5.22 Feedback Class Reference	47
5.22.1 Detailed Description	47
5.22.2 Member Function Documentation	48
5.22.2.1 get()	48
5.22.2.2 init()	48
5.22.2.3 is_on_target()	48
5.22.2.4 set_limits()	48
5.22.2.5 update()	49
5.23 FeedForward Class Reference	49
5.23.1 Detailed Description	50
5.23.2 Constructor & Destructor Documentation	50
5.23.2.1 FeedForward()	50
5.23.3 Member Function Documentation	50
5.23.3.1 calculate()	50
5.24 FeedForward::ff_config_t Struct Reference	51
5.24.1 Detailed Description	51
5.24.2 Member Data Documentation	51
5.24.2.1 kA	51
5.24.2.2 kG	52
5.24.2.3 kS	52
5.24.2.4 kV	52
5.25 Filter Class Reference	52
5.25.1 Detailed Description	52
5.25.2 Member Function Documentation	53
5.25.2.1 add_entry()	53
5.25.2.2 get_value()	53
5.26 Flywheel Class Reference	53
5.26.1 Detailed Description	54
5.26.2 Constructor & Destructor Documentation	54

5.26.2.1 Flywheel()	54
5.26.3 Member Function Documentation	54
5.26.3.1 get_motors()	54
5.26.3.2 get_target()	54
5.26.3.3 getRPM()	55
5.26.3.4 is_on_target()	55
5.26.3.5 Page()	55
5.26.3.6 spin_manual()	55
5.26.3.7 spin_rpm()	56
5.26.3.8 SpinRpmCmd()	56
5.26.3.9 stop()	56
5.26.3.10 WaitUntilUpToSpeedCmd()	56
5.26.4 Friends And Related Symbol Documentation	57
5.26.4.1 spinRPMTask	57
5.27 FlywheelPage Class Reference	57
5.27.1 Member Function Documentation	57
5.27.1.1 draw()	57
5.27.1.2 update()	58
5.28 FlywheelStopCommand Class Reference	58
5.28.1 Detailed Description	59
5.28.2 Constructor & Destructor Documentation	59
5.28.2.1 FlywheelStopCommand()	59
5.28.3 Member Function Documentation	59
5.28.3.1 run()	59
5.29 FlywheelStopMotorsCommand Class Reference	59
5.29.1 Detailed Description	60
5.29.2 Constructor & Destructor Documentation	60
5.29.2.1 FlywheelStopMotorsCommand()	60
5.29.3 Member Function Documentation	60
5.29.3.1 run()	60
5.30 FlywheelStopNonTasksCommand Class Reference	61
5.30.1 Detailed Description	61
5.31 FunctionCommand Class Reference	62
5.31.1 Detailed Description	62
5.31.2 Member Function Documentation	63
5.31.2.1 run()	63
5.32 FunctionCondition Class Reference	63
5.32.1 Detailed Description	63
5.32.2 Member Function Documentation	64
5.32.2.1 test()	64
5.33 screen::FunctionPage Class Reference	64
5.33.1 Detailed Description	64

5.33.2 Constructor & Destructor Documentation	64
5.33.2.1 FunctionPage()	64
5.33.3 Member Function Documentation	65
5.33.3.1 draw()	65
5.33.3.2 update()	65
5.34 GenericAuto Class Reference	65
5.34.1 Detailed Description	66
5.34.2 Member Function Documentation	66
5.34.2.1 add()	66
5.34.2.2 add_async()	66
5.34.2.3 add_delay()	66
5.34.2.4 run()	67
5.35 GraphDrawer Class Reference	67
5.35.1 Constructor & Destructor Documentation	67
5.35.1.1 GraphDrawer()	67
5.35.2 Member Function Documentation	68
5.35.2.1 add_samples() [1/2]	68
5.35.2.2 add_samples() [2/2]	68
5.35.2.3 draw()	68
5.36 PurePursuit::hermite_point Struct Reference	69
5.36.1 Detailed Description	69
5.37 IfTimePassed Class Reference	69
5.37.1 Detailed Description	70
5.37.2 Member Function Documentation	70
5.37.2.1 test()	70
5.38 InOrder Class Reference	70
5.38.1 Detailed Description	71
5.38.2 Member Function Documentation	71
5.38.2.1 on_timeout()	71
5.38.2.2 run()	71
5.39 screen::LabelConfig Struct Reference	72
5.40 Lift< T > Class Template Reference	72
5.40.1 Detailed Description	72
5.40.2 Constructor & Destructor Documentation	73
5.40.2.1 Lift()	73
5.40.3 Member Function Documentation	73
5.40.3.1 control_continuous()	73
5.40.3.2 control_manual()	73
5.40.3.3 control_setpoints()	74
5.40.3.4 get_async()	74
5.40.3.5 get_setpoint()	74
5.40.3.6 hold()	74

5.40.3.7 home()	75
5.40.3.8 set_async()	75
5.40.3.9 set_position()	75
5.40.3.10 set_sensor_function()	75
5.40.3.11 set_sensor_reset()	76
5.40.3.12 set_setpoint()	76
5.41 Lift< T >::lift_cfg_t Struct Reference	76
5.41.1 Detailed Description	76
5.42 Logger Class Reference	77
5.42.1 Detailed Description	77
5.42.2 Constructor & Destructor Documentation	77
5.42.2.1 Logger()	77
5.42.3 Member Function Documentation	78
5.42.3.1 Log() [1/2]	78
5.42.3.2 Log() [2/2]	78
5.42.3.3 Logf() [1/2]	78
5.42.3.4 Logf() [2/2]	79
5.42.3.5 LogIn()	79
5.42.3.6 LogIn() [2/2]	79
5.43 MotionController::m_profile_cfg_t Struct Reference	79
5.43.1 Detailed Description	80
5.44 Mat2 Struct Reference	80
5.45 StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage Class Reference	80
5.45.1 Detailed Description	81
5.45.2 Constructor & Destructor Documentation	81
5.45.2.1 MaybeMessage()	81
5.45.3 Member Function Documentation	81
5.45.3.1 has_message()	81
5.45.3.2 message()	82
5.46 MecanumDrive Class Reference	82
5.46.1 Detailed Description	82
5.46.2 Constructor & Destructor Documentation	82
5.46.2.1 MecanumDrive()	82
5.46.3 Member Function Documentation	83
5.46.3.1 auto_drive()	83
5.46.3.2 auto_turn()	83
5.46.3.3 drive()	84
5.46.3.4 drive_raw()	85
5.47 MecanumDrive::mecanumdrive_config_t Struct Reference	85
5.47.1 Detailed Description	85
5.48 motion_t Struct Reference	85
5.48.1 Detailed Description	86

5.49 MotionController Class Reference	86
5.49.1 Detailed Description	87
5.49.2 Constructor & Destructor Documentation	87
5.49.2.1 MotionController()	87
5.49.3 Member Function Documentation	88
5.49.3.1 get()	88
5.49.3.2 get_motion()	88
5.49.3.3 init()	88
5.49.3.4 is_on_target()	88
5.49.3.5 set_limits()	89
5.49.3.6 tune_feedforward()	89
5.49.3.7 update()	90
5.50 MotionControllerPage Class Reference	90
5.50.1 Member Function Documentation	90
5.50.1.1 draw()	90
5.50.1.2 update()	91
5.51 MovingAverage Class Reference	91
5.51.1 Detailed Description	92
5.51.2 Constructor & Destructor Documentation	92
5.51.2.1 MovingAverage() [1/2]	92
5.51.2.2 MovingAverage() [2/2]	92
5.51.3 Member Function Documentation	92
5.51.3.1 add_entry()	92
5.51.3.2 get_size()	93
5.51.3.3 get_value()	93
5.52 Odometry3Wheel Class Reference	93
5.52.1 Detailed Description	95
5.52.2 Constructor & Destructor Documentation	95
5.52.2.1 Odometry3Wheel()	95
5.52.3 Member Function Documentation	95
5.52.3.1 tune()	95
5.52.3.2 update()	96
5.53 Odometry3Wheel::odometry3wheel_cfg_t Struct Reference	96
5.53.1 Detailed Description	96
5.53.2 Member Data Documentation	97
5.53.2.1 off_axis_center_dist	97
5.53.2.2 wheel_diam	97
5.53.2.3 wheelbase_dist	97
5.54 OdometryBase Class Reference	97
5.54.1 Detailed Description	98
5.54.2 Constructor & Destructor Documentation	98
5.54.2.1 OdometryBase()	98

5.54.3 Member Function Documentation	99
5.54.3.1 background_task()	99
5.54.3.2 end_async()	99
5.54.3.3 get_accel()	99
5.54.3.4 get_angular_accel_deg()	99
5.54.3.5 get_angular_speed_deg()	100
5.54.3.6 get_position()	100
5.54.3.7 get_speed()	100
5.54.3.8 pos_diff()	100
5.54.3.9 rot_diff()	101
5.54.3.10 set_position()	101
5.54.3.11 smallest_angle()	101
5.54.3.12 update()	102
5.54.4 Member Data Documentation	102
5.54.4.1 accel	102
5.54.4.2 ang_accel_deg	102
5.54.4.3 ang_speed_deg	102
5.54.4.4 current_pos	103
5.54.4.5 handle	103
5.54.4.6 mut	103
5.54.4.7 speed	103
5.54.4.8 zero_pos	103
5.55 screen::OdometryPage Class Reference	103
5.55.1 Detailed Description	104
5.55.2 Constructor & Destructor Documentation	104
5.55.2.1 OdometryPage()	104
5.55.3 Member Function Documentation	104
5.55.3.1 draw()	104
5.55.3.2 update()	105
5.56 OdometryTank Class Reference	105
5.56.1 Detailed Description	106
5.56.2 Constructor & Destructor Documentation	106
5.56.2.1 OdometryTank() [1/3]	106
5.56.2.2 OdometryTank() [2/3]	107
5.56.2.3 OdometryTank() [3/3]	107
5.56.3 Member Function Documentation	108
5.56.3.1 set_position()	108
5.56.3.2 update()	108
5.57 OdomSetPosition Class Reference	108
5.57.1 Detailed Description	109
5.57.2 Constructor & Destructor Documentation	109
5.57.2.1 OdomSetPosition()	109

5.57.3 Member Function Documentation	110
5.57.3.1 run()	110
5.58 OrCondition Class Reference	110
5.58.1 Member Function Documentation	110
5.58.1.1 test()	110
5.59 screen::Page Class Reference	111
5.59.1 Detailed Description	111
5.59.2 Member Function Documentation	111
5.59.2.1 draw()	111
5.59.2.2 update()	112
5.60 Parallel Class Reference	112
5.60.1 Detailed Description	113
5.60.2 Member Function Documentation	113
5.60.2.1 on_timeout()	113
5.60.2.2 run()	113
5.61 parallel_runner_info Struct Reference	114
5.62 PurePursuit::Path Class Reference	114
5.62.1 Detailed Description	114
5.62.2 Constructor & Destructor Documentation	114
5.62.2.1 Path()	114
5.62.3 Member Function Documentation	115
5.62.3.1 get_points()	115
5.62.3.2 get_radius()	115
5.62.3.3 is_valid()	115
5.63 PID Class Reference	115
5.63.1 Detailed Description	116
5.63.2 Member Enumeration Documentation	116
5.63.2.1 ERROR_TYPE	116
5.63.3 Constructor & Destructor Documentation	116
5.63.3.1 PID()	116
5.63.4 Member Function Documentation	117
5.63.4.1 get()	117
5.63.4.2 get_error()	117
5.63.4.3 get_sensor_val()	117
5.63.4.4 get_target()	118
5.63.4.5 init()	118
5.63.4.6 is_on_target()	118
5.63.4.7 reset()	118
5.63.4.8 set_limits()	119
5.63.4.9 set_target()	119
5.63.4.10 update()	119
5.63.5 Member Data Documentation	120

5.63.5.1 config	120
5.64 PID::pid_config_t Struct Reference	120
5.64.1 Detailed Description	120
5.64.2 Member Data Documentation	120
5.64.2.1 error_method	120
5.64.2.2 on_target_time	121
5.65 PIDFF Class Reference	121
5.65.1 Member Function Documentation	121
5.65.1.1 get()	121
5.65.1.2 init()	122
5.65.1.3 is_on_target()	123
5.65.1.4 set_limits()	123
5.65.1.5 set_target()	123
5.65.1.6 update() [1/2]	124
5.65.1.7 update() [2/2]	124
5.66 screen::PIDPage Class Reference	125
5.66.1 Detailed Description	125
5.66.2 Constructor & Destructor Documentation	125
5.66.2.1 PIDPage()	125
5.66.3 Member Function Documentation	126
5.66.3.1 draw()	126
5.66.3.2 update()	126
5.67 plm_frame_t Struct Reference	126
5.68 plm_packet_t Struct Reference	127
5.69 plm_plane_t Struct Reference	127
5.70 plm_samples_t Struct Reference	127
5.71 point_t Struct Reference	127
5.71.1 Detailed Description	128
5.71.2 Member Function Documentation	128
5.71.2.1 dist()	128
5.71.2.2 operator+()	128
5.71.2.3 operator-()	129
5.72 pose_t Struct Reference	129
5.72.1 Detailed Description	130
5.73 PurePursuitCommand Class Reference	130
5.73.1 Detailed Description	130
5.73.2 Constructor & Destructor Documentation	131
5.73.2.1 PurePursuitCommand()	131
5.73.3 Member Function Documentation	131
5.73.3.1 on_timeout()	131
5.73.3.2 run()	131
5.74 Rect Struct Reference	131

5.75 RepeatUntil Class Reference	132
5.75.1 Constructor & Destructor Documentation	133
5.75.1.1 RepeatUntil() [1/2]	133
5.75.1.2 RepeatUntil() [2/2]	134
5.75.2 Member Function Documentation	134
5.75.2.1 on_timeout()	134
5.75.2.2 run()	134
5.76 robot_specs_t Struct Reference	135
5.76.1 Detailed Description	135
5.77 screen::ScreenData Struct Reference	135
5.77.1 Detailed Description	136
5.78 screen::ScreenRect Struct Reference	136
5.79 Serializer Class Reference	136
5.79.1 Detailed Description	137
5.79.2 Constructor & Destructor Documentation	137
5.79.2.1 Serializer()	137
5.79.3 Member Function Documentation	137
5.79.3.1 bool_or()	137
5.79.3.2 double_or()	137
5.79.3.3 int_or()	138
5.79.3.4 save_to_disk()	138
5.79.3.5 set_bool()	138
5.79.3.6 set_double()	139
5.79.3.7 set_int()	139
5.79.3.8 set_string()	139
5.79.3.9 string_or()	139
5.80 screen::SizedWidget Struct Reference	140
5.81 SliderCfg Struct Reference	140
5.82 screen::SliderConfig Struct Reference	140
5.83 screen::SliderWidget Class Reference	141
5.83.1 Detailed Description	141
5.83.2 Constructor & Destructor Documentation	141
5.83.2.1 SliderWidget()	141
5.83.3 Member Function Documentation	141
5.83.3.1 update()	141
5.84 SpinRPMCommand Class Reference	142
5.84.1 Detailed Description	143
5.84.2 Constructor & Destructor Documentation	143
5.84.2.1 SpinRPMCommand()	143
5.84.3 Member Function Documentation	143
5.84.3.1 run()	143
5.85 PurePursuit::spline Struct Reference	144

5.85.1 Detailed Description	144
5.86 StateMachine< System, IDType, Message, delay_ms, do_log >::State Struct Reference	144
5.86.1 Detailed Description	144
5.87 StateMachine< System, IDType, Message, delay_ms, do_log > Class Template Reference	145
5.87.1 Detailed Description	145
5.87.2 Constructor & Destructor Documentation	146
5.87.2.1 StateMachine()	146
5.87.3 Member Function Documentation	146
5.87.3.1 current_state()	146
5.87.3.2 send_message()	146
5.88 screen::StatsPage Class Reference	147
5.88.1 Detailed Description	147
5.88.2 Constructor & Destructor Documentation	147
5.88.2.1 StatsPage()	147
5.88.3 Member Function Documentation	147
5.88.3.1 draw()	147
5.88.3.2 update()	148
5.89 TakeBackHalf Class Reference	148
5.89.1 Detailed Description	149
5.89.2 Member Function Documentation	149
5.89.2.1 get()	149
5.89.2.2 init()	149
5.89.2.3 is_on_target()	149
5.89.2.4 set_limits()	150
5.89.2.5 update()	150
5.90 TankDrive Class Reference	150
5.90.1 Detailed Description	152
5.90.2 Member Enumeration Documentation	152
5.90.2.1 BrakeType	152
5.90.3 Constructor & Destructor Documentation	152
5.90.3.1 TankDrive()	152
5.90.4 Member Function Documentation	152
5.90.4.1 drive_arcade()	152
5.90.4.2 drive_forward() [1/2]	153
5.90.4.3 drive_forward() [2/2]	154
5.90.4.4 drive_tank()	154
5.90.4.5 drive_tank_raw()	155
5.90.4.6 drive_to_point() [1/2]	155
5.90.4.7 drive_to_point() [2/2]	156
5.90.4.8 modify_inputs()	157
5.90.4.9 pure_pursuit() [1/2]	157
5.90.4.10 pure_pursuit() [2/2]	158

5.90.4.11 reset_auto()	159
5.90.4.12 stop()	159
5.90.4.13 turn_degrees() [1/2]	159
5.90.4.14 turn_degrees() [2/2]	160
5.90.4.15 turn_to_heading() [1/2]	160
5.90.4.16 turn_to_heading() [2/2]	161
5.91 screen::TextConfig Struct Reference	162
5.92 TimesTestedCondition Class Reference	162
5.92.1 Member Function Documentation	162
5.92.1.1 test()	162
5.93 trapezoid_profile_segment_t Struct Reference	162
5.93.1 Detailed Description	163
5.94 TrapezoidProfile Class Reference	163
5.94.1 Detailed Description	164
5.94.2 Constructor & Destructor Documentation	164
5.94.2.1 TrapezoidProfile()	164
5.94.3 Member Function Documentation	165
5.94.3.1 calculate()	165
5.94.3.2 calculate_time_based()	165
5.94.3.3 get_movement_time()	165
5.94.3.4 set_accel()	166
5.94.3.5 set_endpts()	166
5.94.3.6 set_max_v()	166
5.94.3.7 set_vel_endpts()	166
5.95 TurnDegreesCommand Class Reference	167
5.95.1 Detailed Description	168
5.95.2 Constructor & Destructor Documentation	168
5.95.2.1 TurnDegreesCommand()	168
5.95.3 Member Function Documentation	168
5.95.3.1 on_timeout()	168
5.95.3.2 run()	168
5.96 TurnToHeadingCommand Class Reference	169
5.96.1 Detailed Description	169
5.96.2 Constructor & Destructor Documentation	169
5.96.2.1 TurnToHeadingCommand()	169
5.96.3 Member Function Documentation	170
5.96.3.1 on_timeout()	170
5.96.3.2 run()	170
5.97 Vector2D Class Reference	170
5.97.1 Detailed Description	171
5.97.2 Constructor & Destructor Documentation	171
5.97.2.1 Vector2D() [1/2]	171

5.97.2.2 Vector2D() [2/2]	171
5.97.3 Member Function Documentation	172
5.97.3.1 get_dir()	172
5.97.3.2 get_mag()	172
5.97.3.3 get_x()	172
5.97.3.4 get_y()	172
5.97.3.5 normalize()	173
5.97.3.6 operator*()	173
5.97.3.7 operator+()	173
5.97.3.8 operator-()	173
5.97.3.9 point()	174
5.98 VideoPlayer Class Reference	174
5.98.1 Member Function Documentation	174
5.98.1.1 draw()	174
5.98.1.2 update()	175
5.99 WaitUntilCondition Class Reference	175
5.99.1 Detailed Description	176
5.99.2 Member Function Documentation	176
5.99.2.1 run()	176
5.100 WaitUntilUpToSpeedCommand Class Reference	176
5.100.1 Detailed Description	177
5.100.2 Constructor & Destructor Documentation	177
5.100.2.1 WaitUntilUpToSpeedCommand()	177
5.100.3 Member Function Documentation	178
5.100.3.1 run()	178
5.101 screen::WidgetConfig Struct Reference	178
5.102 screen::WidgetPage Class Reference	179
5.102.1 Member Function Documentation	179
5.102.1.1 draw()	179
5.102.1.2 update()	179
6 File Documentation	181
6.1 robot_specs.h	181
6.2 custom_encoder.h	181
6.3 flywheel.h	182
6.4 pl_mpeg.h	182
6.5 video.h	232
6.6 layout.h	232
6.7 lift.h	232
6.8 mecanum_drive.h	235
6.9 odometry_3wheel.h	236
6.10 odometry_base.h	236

6.11 odometry_tank.h	237
6.12 screen.h	238
6.13 tank_drive.h	240
6.14 auto_chooser.h	242
6.15 auto_command.h	242
6.16 basic_command.h	245
6.17 command_controller.h	245
6.18 delay_command.h	246
6.19 drive_commands.h	246
6.20 flywheel_commands.h	248
6.21 bang_bang.h	249
6.22 feedback_base.h	249
6.23 feedforward.h	249
6.24 motion_controller.h	250
6.25 pid.h	251
6.26 pidff.h	252
6.27 take_back_half.h	252
6.28 trapezoid_profile.h	253
6.29 generic_auto.h	253
6.30 geometry.h	254
6.31 graph_drawer.h	255
6.32 logger.h	256
6.33 math_util.h	256
6.34 moving_average.h	257
6.35 pure_pursuit.h	258
6.36 serializer.h	259
6.37 state_machine.h	260
6.38 vector2d.h	261
Index	263

Chapter 1

Core

This is the host repository for the custom VEX libraries used by the RIT VEXU team

Automatically updated documentation is available at [here](#). There is also a downloadable [reference manual](#).

1.1 Getting Started

In order to simply use this repo, you can either clone it into your VEXcode project folder, or download the .zip and place it into a core/ subfolder. Then follow the instructions for setting up compilation at [Wiki/BuildSystem](#)

If you wish to contribute, follow the instructions at [Wiki/ProjectSetup](#)

1.2 Features

Here is the current feature list this repo provides:

Subsystems (See [Wiki/Subsystems](#)):

- Tank drivetrain (user control / autonomous)
- Mecanum drivetrain (user control / autonomous)
- Odometry
- [Flywheel](#)
- [Lift](#)
- Custom encoders

Utilities (See [Wiki/Utilites](#)):

- [PID](#) controller
- [FeedForward](#) controller
- Trapezoidal motion profile controller
- Pure Pursuit
- Generic auto program builder
- Auto program UI selector
- Mathematical classes ([Vector2D](#), Moving Average)

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

AutoCommand	18
Async	14
BasicSolenoidSet	22
BasicSpinCommand	23
BasicStopCommand	25
Branch	27
DelayCommand	36
DriveForwardCommand	38
DriveStopCommand	40
DriveToPointCommand	42
FlywheelStopCommand	58
FlywheelStopMotorsCommand	59
FlywheelStopNonTasksCommand	61
FunctionCommand	62
InOrder	70
OdomSetPosition	108
Parallel	112
PurePursuitCommand	130
RepeatUntil	132
SpinRPMCommand	142
TurnDegreesCommand	167
TurnToHeadingCommand	169
WaitUntilCondition	175
WaitUntilUpToSpeedCommand	176
screen::ButtonConfig	28
screen::ButtonWidget	28
screen::CheckboxConfig	30
CommandController	30
Condition	33
AndCondition	13
FunctionCondition	63
IfTimePassed	69
OrCondition	110
TimesTestedCondition	162
vex::encoder	

CustomEncoder	34
AutoChooser::entry_t	44
Feedback	47
BangBang	20
MotionController	86
PID	115
PIDFF	121
TakeBackHalf	148
FeedForward	49
FeedForward::ff_config_t	51
Filter	52
ExponentialMovingAverage	45
MovingAverage	91
Flywheel	53
GenericAuto	65
GraphDrawer	67
PurePursuit::hermite_point	69
screen::LabelConfig	72
Lift< T >	72
Lift< T >::lift_cfg_t	76
Logger	77
MotionController::m_profile_cfg_t	79
Mat2	80
StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage	80
MecanumDrive	82
MecanumDrive::mecanumdrive_config_t	85
motion_t	85
Odometry3Wheel::odometry3wheel_cfg_t	96
OdometryBase	97
Odometry3Wheel	93
OdometryTank	105
screen::Page	111
AutoChooser	15
FlywheelPage	57
MotionControllerPage	90
VideoPlayer	174
screen::FunctionPage	64
screen::OdometryPage	103
screen::PIDPage	125
screen::StatsPage	147
screen::WidgetPage	179
parallel_runner_info	114
PurePursuit::Path	114
PID::pid_config_t	120
plm_frame_t	126
plm_packet_t	127
plm_plane_t	127
plm_samples_t	127
point_t	127
pose_t	129
Rect	131
robot_specs_t	135
screen::ScreenData	135
screen::ScreenRect	136
Serializer	136
screen::SizedWidget	140
SliderCfg	140

screen::SliderConfig	140
screen::SliderWidget	141
PurePursuit::spline	144
StateMachine< System, IDType, Message, delay_ms, do_log >::State	144
StateMachine< System, IDType, Message, delay_ms, do_log >	145
TankDrive	150
screen::TextConfig	162
trapezoid_profile_segment_t	162
TrapezoidProfile	163
Vector2D	170
screen::WidgetConfig	178

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

AndCondition	13
Async	
Async runs a command asynchronously will simply let it go and never look back THIS HAS A VERY NICHE USE CASE. THINK ABOUT IF YOU REALLY NEED IT	14
AutoChooser	15
AutoCommand	18
BangBang	20
BasicSolenoidSet	22
BasicSpinCommand	23
BasicStopCommand	25
Branch	
Branch chooses from multiple options at runtime. the function decider returns an index into the choices vector If you wish to make no choice and skip this section, return NO_CHOICE; any choice that is out of bounds set to NO_CHOICE	27
screen::ButtonConfig	28
screen::ButtonWidget	
Widget that does something when you tap it. The function is only called once when you first tap it	28
screen::CheckboxConfig	30
CommandController	30
Condition	33
CustomEncoder	34
DelayCommand	36
DriveForwardCommand	38
DriveStopCommand	40
DriveToPointCommand	42
AutoChooser::entry_t	44
ExponentialMovingAverage	45
Feedback	47
FeedForward	49
FeedForward::ff_config_t	51
Filter	52
Flywheel	53
FlywheelPage	57
FlywheelStopCommand	58
FlywheelStopMotorsCommand	59

FlywheelStopNonTasksCommand	61
FunctionCommand	62
FunctionCondition	
FunctionCondition	is a quick and dirty Condition to wrap some expression that should be evaluated at runtime
	63
screen::FunctionPage	
Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything	64
GenericAuto	65
GraphDrawer	67
PurePursuit::hermite_point	69
IfTimePassed	
IfTimePassed	tests based on time since the command controller was constructed. Returns true if elapsed time > time_s
	69
InOrder	
InOrder	runs its commands sequentially then continues. How to handle timeout in this case.
Automatically set it to sum of commands timeouts?	70
screen::LabelConfig	72
Lift< T >	72
Lift< T >::lift_cfg_t	76
Logger	
Class to simplify writing to files	77
MotionController::m_profile_cfg_t	79
Mat2	80
StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage	
MaybeMessage	a message of Message type or nothing
MaybeMessage	m = {}; // empty
MaybeMessage	m = Message::EnumField1
MecanumDrive	82
MecanumDrive::mecanumdrive_config_t	85
motion_t	85
MotionController	86
MotionControllerPage	90
MovingAverage	91
Odometry3Wheel	93
Odometry3Wheel::odometry3wheel_cfg_t	96
OdometryBase	97
screen::OdometryPage	
Page	that shows odometry position and rotation and a map (if an sd card with the file is on)
	103
OdometryTank	105
OdomSetPosition	108
OrCondition	110
screen::Page	
Page	describes one part of the screen slideshow
	111
Parallel	
Parallel	runs multiple commands in parallel and waits for all to finish before continuing. if none finish before this command's timeout, it will call on_timeout on all children continue
	112
parallel_runner_info	114
PurePursuit::Path	114
PID	115
PID::pid_config_t	120
PIDFF	121
screen::PIDPage	
PIDPage	provides a way to tune a pid controller on the screen
	125
plm_frame_t	126
plm_packet_t	127
plm_plane_t	127
plm_samples_t	127
point_t	127

pose_t	129
PurePursuitCommand	130
Rect	131
RepeatUntil	132
robot_specs_t	135
screen::ScreenData Holds the data that will be passed to the screen thread you probably shouldnt have to use it	135
screen::ScreenRect	136
Serializer Serializes Arbitrary data to a file on the SD Card	136
screen::SizedWidget	140
SliderCfg	140
screen::SliderConfig	140
screen::SliderWidget Widget that updates a double value. Updates by reference so watch out for race conditions cuz the screen stuff lives on another thread	141
SpinRPMCommand	142
PurePursuit::spline	144
StateMachine< System, IDType, Message, delay_ms, do_log >::State	144
StateMachine< System, IDType, Message, delay_ms, do_log > State Machine :)))))) A fun fun way of controlling stateful subsystems - used in the 2023-2024 Over Under game for our overly complex intake-cata subsystem (see there for an example) The statemachine runs in a background thread and a user thread can interact with it through current_state and send_message	145
screen::StatsPage Draws motor stats and battery stats to the screen	147
TakeBackHalf A velocity controller	148
TankDrive	150
screen::TextConfig	162
TimesTestedCondition	162
trapezoid_profile_segment_t	162
TrapezoidProfile	163
TurnDegreesCommand	167
TurnToHeadingCommand	169
Vector2D	170
VideoPlayer	174
WaitUntilCondition Waits until the condition is true	175
WaitUntilUpToSpeedCommand	176
screen::WidgetConfig	178
screen::WidgetPage	179

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

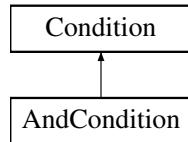
include/robot_specs.h	181
include/subsystems/custom_encoder.h	181
include/subsystems/flywheel.h	182
include/subsystems/layout.h	232
include/subsystems/lift.h	232
include/subsystems/mecanum_drive.h	235
include/subsystems/screen.h	238
include/subsystems/tank_drive.h	240
include/subsystems/fun/pl_mpeg.h	182
include/subsystems/fun/video.h	232
include/subsystems/odometry/odometry_3wheel.h	236
include/subsystems/odometry/odometry_base.h	236
include/subsystems/odometry/odometry_tank.h	237
include/utils/auto_chooser.h	242
include/utils/generic_auto.h	253
include/utils/geometry.h	254
include/utils/graph_drawer.h	255
include/utils/logger.h	256
include/utils/math_util.h	256
include/utils/moving_average.h	257
include/utils/pure_pursuit.h	258
include/utils/serializer.h	259
include/utils/state_machine.h	260
include/utils/vector2d.h	261
include/utils/command_structure/auto_command.h	242
include/utils/command_structure/basic_command.h	245
include/utils/command_structure/command_controller.h	245
include/utils/command_structure/delay_command.h	246
include/utils/command_structure/drive_commands.h	246
include/utils/command_structure/flywheel_commands.h	248
include/utils/controls/bang_bang.h	249
include/utils/controls/feedback_base.h	249
include/utils/controls/feedforward.h	249
include/utils/controls/motion_controller.h	250
include/utils/controls/pid.h	251
include/utils/controls/pidff.h	252
include/utils/controls/take_back_half.h	252
include/utils/controls/trapezoid_profile.h	253

Chapter 5

Class Documentation

5.1 AndCondition Class Reference

Inheritance diagram for AndCondition:



Public Member Functions

- **AndCondition** ([Condition](#) *A, [Condition](#) *B)
- bool [test](#) () override

Public Member Functions inherited from [Condition](#)

- [Condition](#) * **Or** ([Condition](#) *b)
- [Condition](#) * **And** ([Condition](#) *b)

5.1.1 Member Function Documentation

5.1.1.1 [test\(\)](#)

```
bool AndCondition::test ( ) [inline], [override], [virtual]
```

Implements [Condition](#).

The documentation for this class was generated from the following file:

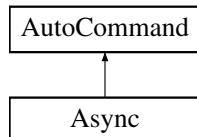
- src/utils/command_structure/auto_command.cpp

5.2 Async Class Reference

[Async](#) runs a command asynchronously will simply let it go and never look back THIS HAS A VERY NICHE USE CASE. THINK ABOUT IF YOU REALLY NEED IT.

```
#include <auto_command.h>
```

Inheritance diagram for Async:



Public Member Functions

- **Async** ([AutoCommand](#) *cmd)
- bool [run](#) () override

Public Member Functions inherited from [AutoCommand](#)

- virtual void [on_timeout](#) ()
- [AutoCommand](#) * [withTimeout](#) (double t_seconds)
- [AutoCommand](#) * [withCancelCondition](#) ([Condition](#) *true_to_end)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- double [timeout_seconds](#) = default_timeout
- [Condition](#) * [true_to_end](#) = nullptr

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default_timeout](#) = 10.0

5.2.1 Detailed Description

[Async](#) runs a command asynchronously will simply let it go and never look back THIS HAS A VERY NICHE USE CASE. THINK ABOUT IF YOU REALLY NEED IT.

5.2.2 Member Function Documentation

5.2.2.1 run()

```
bool Async::run ( ) [override], [virtual]
```

Executes the command Overridden by child classes

Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

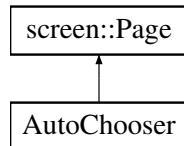
The documentation for this class was generated from the following files:

- [include/utils/command_structure/auto_command.h](#)
- [src/utils/command_structure/auto_command.cpp](#)

5.3 AutoChooser Class Reference

```
#include <auto_chooser.h>
```

Inheritance diagram for AutoChooser:



Classes

- struct [entry_t](#)

Public Member Functions

- [AutoChooser](#) (`std::vector< std::string > paths, size_t def=0)`
- void [update](#) (`bool was_pressed, int x, int y)`
collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this Page (only drawn page gets touch updates))
- void [draw](#) (`vex::brain::lcd &, bool first_draw, unsigned int frame_number)`
draw stored data to the screen (runs at 10 hz and only runs if this page is in front)
- `size_t get_choice ()`

Protected Attributes

- `size_t choice`
- `std::vector< entry_t > list`

Static Protected Attributes

- static const size_t **width** = 380
- static const size_t **height** = 220

5.3.1 Detailed Description

Autochooser is a utility to make selecting robot autonomous programs easier source: RIT VexU Wiki During a season, we usually code between 4 and 6 autonomous programs. Most teams will change their entire robot program as a way of choosing autonomy but this may cause issues if you have an emergency patch to upload during a competition. This class was built as a way of using the robot screen to list autonomous programs, and the touchscreen to select them.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 AutoChooser()

```
AutoChooser::AutoChooser (
    std::vector< std::string > paths,
    size_t def = 0 )
```

Initialize the auto-chooser. This class places a choice menu on the brain screen, so the driver can choose which autonomous to run.

Parameters

<i>brain</i>	the brain on which to draw the selection boxes
--------------	--

5.3.3 Member Function Documentation

5.3.3.1 draw()

```
void AutoChooser::draw (
    vex::brain::lcd & screen,
    bool first_draw,
    unsigned int frame_number ) [virtual]
```

draw stored data to the screen (runs at 10 hz and only runs if this page is in front)

Parameters

<i>first_draw</i>	true if we just switched to this page
<i>frame_number</i>	frame of drawing we are on (basically an animation tick)

Reimplemented from [screen::Page](#).

5.3.3.2 get_choice()

```
size_t AutoChooser::get_choice ( )
```

Get the currently selected auto choice

Returns

the identifier to the auto path

Return the selected autonomous

5.3.3.3 update()

```
void AutoChooser::update (
    bool was_pressed,
    int x,
    int y ) [virtual]
```

collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this Page (only drawn page gets touch updates))

Parameters

<i>was_pressed</i>	true if the screen has been pressed
<i>x</i>	x position of screen press (if the screen was pressed)
<i>y</i>	y position of screen press (if the screen was pressed)

Reimplemented from [screen::Page](#).

5.3.4 Member Data Documentation

5.3.4.1 choice

```
size_t AutoChooser::choice [protected]
```

the current choice of auto

5.3.4.2 list

```
std::vector<entry\_t> AutoChooser::list [protected]
```

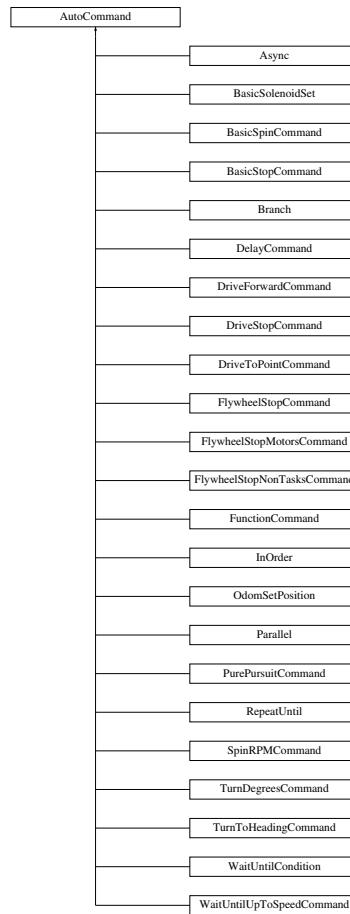
< a list of all possible auto choices

The documentation for this class was generated from the following files:

- [include/utils/auto_chooser.h](#)
- [src/utils/auto_chooser.cpp](#)

5.4 AutoCommand Class Reference

Inheritance diagram for AutoCommand:



Public Member Functions

- `virtual bool run ()`
- `virtual void on_timeout ()`
- `AutoCommand * withTimeout (double t_seconds)`
- `AutoCommand * withCancelCondition (Condition *true_to_end)`

Public Attributes

- `double timeout_seconds = default_timeout`
- `Condition * true_to_end = nullptr`

Static Public Attributes

- `static constexpr double default_timeout = 10.0`

5.4.1 Member Function Documentation

5.4.1.1 on_timeout()

```
virtual void AutoCommand::on_timeout ( ) [inline], [virtual]
```

What to do if we timeout instead of finishing. timeout is specified by the timeout seconds in the constructor

Reimplemented in [InOrder](#), [Parallel](#), [Branch](#), [RepeatUntil](#), [DriveForwardCommand](#), [TurnDegreesCommand](#), [TurnToHeadingCommand](#), [PurePursuitCommand](#), and [DriveStopCommand](#).

5.4.1.2 run()

```
virtual bool AutoCommand::run ( ) [inline], [virtual]
```

Executes the command Overridden by child classes

Returns

true when the command is finished, false otherwise

Reimplemented in [FunctionCommand](#), [WaitUntilCondition](#), [InOrder](#), [Parallel](#), [Branch](#), [Async](#), [RepeatUntil](#), [BasicSpinCommand](#), [BasicStopCommand](#), [BasicSolenoidSet](#), [DelayCommand](#), [DriveForwardCommand](#), [TurnDegreesCommand](#), [DriveToPointCommand](#), [TurnToHeadingCommand](#), [PurePursuitCommand](#), [DriveStopCommand](#), [OdomSetPosition](#), [SpinRPMCommand](#), [WaitUntilUpToSpeedCommand](#), [FlywheelStopCommand](#), and [FlywheelStopMotorsCommand](#)

5.4.2 Member Data Documentation

5.4.2.1 timeout_seconds

```
double AutoCommand::timeout_seconds = default_timeout
```

How long to run until we cancel this command. If the command is cancelled, [on_timeout\(\)](#) is called to allow any cleanup from the function. If the timeout_seconds <= 0, no timeout will be applied and this command will run forever. A timeout can come in handy for some commands that can not reach the end due to some physical limitation such as

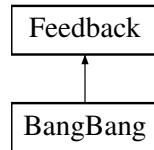
- a drive command hitting a wall and not being able to reach its target
- a command that waits until something is up to speed that never gets up to speed because of battery voltage
- something else...

The documentation for this class was generated from the following file:

- include/utils/command_structure/auto_command.h

5.5 BangBang Class Reference

Inheritance diagram for BangBang:



Public Member Functions

- **BangBang** (double threshold, double low, double high)
- void **init** (double start_pt, double set_pt, double start_vel=0.0, double end_vel=0.0) override
- double **update** (double val) override
- double **get** () override
- void **set_limits** (double lower, double upper) override
- bool **is_on_target** () override

5.5.1 Member Function Documentation

5.5.1.1 get()

```
double BangBang::get () [override], [virtual]
```

Returns

the last saved result from the feedback controller

Implements [Feedback](#).

5.5.1.2 init()

```
void BangBang::init (
    double start_pt,
    double set_pt,
    double start_vel = 0.0,
    double end_vel = 0.0 ) [override], [virtual]
```

Initialize the feedback controller for a movement

Parameters

<i>start_pt</i>	the current sensor value
<i>set_pt</i>	where the sensor value should be
<i>start_vel</i>	Movement starting velocity
<i>end_vel</i>	Movement ending velocity

Implements [Feedback](#).

5.5.1.3 `is_on_target()`

```
bool BangBang::is_on_target ( ) [override], [virtual]
```

Returns

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

5.5.1.4 `set_limits()`

```
void BangBang::set_limits (
    double lower,
    double upper ) [override], [virtual]
```

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied.

Parameters

<i>lower</i>	Upper limit
<i>upper</i>	Lower limit

Implements [Feedback](#).

5.5.1.5 `update()`

```
double BangBang::update (
    double val ) [override], [virtual]
```

Iterate the feedback loop once with an updated sensor value

Parameters

<i>val</i>	value from the sensor
------------	-----------------------

Returns

feedback loop result

Implements [Feedback](#).

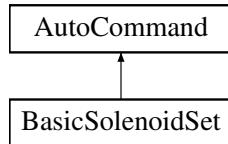
The documentation for this class was generated from the following files:

- include/utils/controls/bang_bang.h
- src/utils/controls/bang_bang.cpp

5.6 BasicSolenoidSet Class Reference

```
#include <basic_command.h>
```

Inheritance diagram for BasicSolenoidSet:



Public Member Functions

- **BasicSolenoidSet** (vex::pneumatics &solenoid, bool setting)
Construct a new BasicSolenoidSet Command.
- bool **run** () override
Runs the BasicSolenoidSet Overrides run command from AutoCommand.

Public Member Functions inherited from AutoCommand

- virtual void **on_timeout** ()
- **AutoCommand** * **withTimeout** (double t_seconds)
- **AutoCommand** * **withCancelCondition** (**Condition** *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double **timeout_seconds** = default_timeout
- **Condition** * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.6.1 Detailed Description

[AutoCommand](#) wrapper class for [BasicSolenoidSet](#) Using the Vex hardware functions

5.6.2 Constructor & Destructor Documentation

5.6.2.1 BasicSolenoidSet()

```
BasicSolenoidSet::BasicSolenoidSet (
    vex::pneumatics & solenoid,
    bool setting )
```

Construct a new [BasicSolenoidSet](#) Command.

Parameters

<i>solenoid</i>	Solenoid being set
<i>setting</i>	Setting of the solenoid in boolean (true,false)

5.6.3 Member Function Documentation**5.6.3.1 run()**

```
bool BasicSolenoidSet::run ( ) [override], [virtual]
```

Runs the [BasicSolenoidSet](#) Overrides run command from [AutoCommand](#).

Returns

True Command runs once

Reimplemented from [AutoCommand](#).

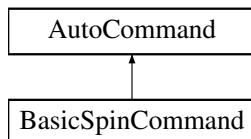
The documentation for this class was generated from the following files:

- include/utils/command_structure/basic_command.h
- src/utils/command_structure/basic_command.cpp

5.7 BasicSpinCommand Class Reference

```
#include <basic_command.h>
```

Inheritance diagram for BasicSpinCommand:

**Public Types**

- enum **type** { **percent** , **voltage** , **veocity** }

Public Member Functions

- [**BasicSpinCommand**](#) (vex::motor &motor, vex::directionType dir, BasicSpinCommand::type setting, double power)

Construct a new BasicSpinCommand.

- bool [**run**](#) () override

Runs the BasicSpinCommand Overrides run from Auto Command.

Public Member Functions inherited from [AutoCommand](#)

- virtual void [on_timeout \(\)](#)
- [AutoCommand * withTimeout](#) (double t_seconds)
- [AutoCommand * withCancelCondition](#) ([Condition](#) *true_to_end)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- double [timeout_seconds](#) = default_timeout
- [Condition * true_to_end](#) = nullptr

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default_timeout](#) = 10.0

5.7.1 Detailed Description

[AutoCommand](#) wrapper class for [BasicSpinCommand](#) using the vex hardware functions

5.7.2 Constructor & Destructor Documentation

5.7.2.1 [BasicSpinCommand\(\)](#)

```
BasicSpinCommand::BasicSpinCommand (
    vex::motor & motor,
    vex::directionType dir,
    BasicSpinCommand::type setting,
    double power )
```

Construct a new [BasicSpinCommand](#).

a BasicMotorSpin Command

Parameters

<i>motor</i>	Motor to spin
<i>direc</i>	Direction of motor spin
<i>setting</i>	Power setting in volts,percentage,velocity
<i>power</i>	Value of desired power
<i>motor</i>	Motor port to spin
<i>dir</i>	Direction for spinning
<i>setting</i>	Power setting in volts,percentage,velocity
<i>power</i>	Value of desired power

5.7.3 Member Function Documentation

5.7.3.1 run()

```
bool BasicSpinCommand::run ( ) [override], [virtual]
```

Runs the [BasicSpinCommand](#) Overrides run from Auto Command.

Run the [BasicSpinCommand](#) Overrides run from Auto Command.

Returns

True [Async](#) running command

True Command runs once

Reimplemented from [AutoCommand](#).

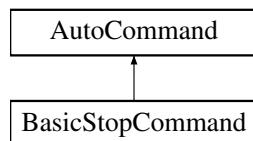
The documentation for this class was generated from the following files:

- include/utils/command_structure/basic_command.h
- src/utils/command_structure/basic_command.cpp

5.8 BasicStopCommand Class Reference

```
#include <basic_command.h>
```

Inheritance diagram for BasicStopCommand:



Public Member Functions

- [BasicStopCommand](#) (vex::motor &motor, vex::brakeType setting)
Construct a new BasicMotorStop Command.
- bool [run \(\)](#) override
Runs the BasicMotorStop Command Overrides run command from [AutoCommand](#).

Public Member Functions inherited from [AutoCommand](#)

- virtual void [on_timeout \(\)](#)
- [AutoCommand * withTimeout](#) (double t_seconds)
- [AutoCommand * withCancelCondition](#) ([Condition](#) *true_to_end)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- double `timeout_seconds` = `default_timeout`
- `Condition * true_to_end` = `nullptr`

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double `default_timeout` = 10.0

5.8.1 Detailed Description

[AutoCommand](#) wrapper class for [BasicStopCommand](#) Using the Vex hardware functions

5.8.2 Constructor & Destructor Documentation

5.8.2.1 [BasicStopCommand\(\)](#)

```
BasicStopCommand::BasicStopCommand (
    vex::motor & motor,
    vex::brakeType setting )
```

Construct a new BasicMotorStop Command.

Construct a BasicMotorStop Command.

Parameters

<code>motor</code>	The motor to stop
<code>setting</code>	The brake setting for the motor
<code>motor</code>	Motor to stop
<code>setting</code>	Braketype setting brake,coast,hold

5.8.3 Member Function Documentation

5.8.3.1 [run\(\)](#)

```
bool BasicStopCommand::run ( ) [override], [virtual]
```

Runs the BasicMotorStop Command Overrides run command from [AutoCommand](#).

Runs the BasicMotorStop command Ovverides run command from [AutoCommand](#).

Returns

True Command runs once

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

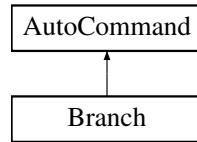
- include/utils/command_structure/basic_command.h
- src/utils/command_structure/basic_command.cpp

5.9 Branch Class Reference

[Branch](#) chooses from multiple options at runtime. the function decider returns an index into the choices vector If you wish to make no choice and skip this section, return NO_CHOICE; any choice that is out of bounds set to NO_CHOICE.

```
#include <auto_command.h>
```

Inheritance diagram for Branch:



Public Member Functions

- **Branch** ([Condition](#) *cond, [AutoCommand](#) *false_choice, [AutoCommand](#) *true_choice)
- bool [run](#) () override
- void [on_timeout](#) () override

Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand](#) * [withTimeout](#) (double t_seconds)
- [AutoCommand](#) * [withCancelCondition](#) ([Condition](#) *true_to_end)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- double [timeout_seconds](#) = default_timeout
- [Condition](#) * [true_to_end](#) = nullptr

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default_timeout](#) = 10.0

5.9.1 Detailed Description

[Branch](#) chooses from multiple options at runtime. the function decider returns an index into the choices vector If you wish to make no choice and skip this section, return NO_CHOICE; any choice that is out of bounds set to NO_CHOICE.

5.9.2 Member Function Documentation

5.9.2.1 on_timeout()

```
void Branch::on_timeout ( ) [override], [virtual]
```

What to do if we timeout instead of finishing. timeout is specified by the timeout seconds in the constructor

Reimplemented from [AutoCommand](#).

5.9.2.2 run()

```
bool Branch::run ( ) [override], [virtual]
```

Executes the command Overridden by child classes

Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command_structure/auto_command.h
- src/utils/command_structure/auto_command.cpp

5.10 screen::ButtonConfig Struct Reference

Public Attributes

- std::function< void()> **onclick**

The documentation for this struct was generated from the following file:

- include/subsystems/screen.h

5.11 screen::ButtonWidget Class Reference

Widget that does something when you tap it. The function is only called once when you first tap it.

```
#include <screen.h>
```

Public Member Functions

- **ButtonWidget** (std::function< void(void)> onpress, `Rect` rect, std::string name)
Create a Button widget.
- **ButtonWidget** (void(*onpress)(), `Rect` rect, std::string name)
Create a Button widget.
- bool **update** (bool was_pressed, int x, int y)
responds to user input
- void **draw** (vex::brain::lcd &, bool first_draw, unsigned int frame_number)
draws the button to the screen

5.11.1 Detailed Description

Widget that does something when you tap it. The function is only called once when you first tap it.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 **ButtonWidget()** [1/2]

```
screen::ButtonWidget::ButtonWidget (
    std::function< void(void)> onpress,
    Rect rect,
    std::string name ) [inline]
```

Create a Button widget.

Parameters

<code>onpress</code>	the function to be called when the button is tapped
<code>rect</code>	the area the button should take up on the screen
<code>name</code>	the label put on the button

5.11.2.2 **ButtonWidget()** [2/2]

```
screen::ButtonWidget::ButtonWidget (
    void(*)() onpress,
    Rect rect,
    std::string name ) [inline]
```

Create a Button widget.

Parameters

<code>onpress</code>	the function to be called when the button is tapped
<code>rect</code>	the area the button should take up on the screen
<code>name</code>	the label put on the button

5.11.3 Member Function Documentation

5.11.3.1 update()

```
bool screen::ButtonWidget::update (
    bool was_pressed,
    int x,
    int y )
```

responds to user input

Parameters

<i>was_pressed</i>	if the screen is pressed
<i>x</i>	x position if the screen was pressed
<i>y</i>	y position if the screen was pressed

Returns

true if the button was pressed

The documentation for this class was generated from the following files:

- include/subsystems/screen.h
- src/subsystems/screen.cpp

5.12 screen::CheckboxConfig Struct Reference

Public Attributes

- std::function< void(bool)> **onupdate**

The documentation for this struct was generated from the following file:

- include/subsystems/screen.h

5.13 CommandController Class Reference

```
#include <command_controller.h>
```

Public Member Functions

- **CommandController ()**
Create an empty [CommandController](#). Add Command with [CommandController::add\(\)](#)
- **CommandController (std::initializer_list< [AutoCommand](#) * > cmd)**
Create a [CommandController](#) with commands pre added. More can be added with [CommandController::add\(\)](#)
- void **add (std::vector< [AutoCommand](#) * > cmd)**
- void **add ([AutoCommand](#) *cmd, double timeout_seconds=10.0)**
- void **add (std::vector< [AutoCommand](#) * > cmd, double timeout_sec)**
- void **add_delay (int ms)**
- void **add_cancel_func (std::function< bool(void)> true_if_cancel)**
add_cancel_func specifies that when this func evaluates to true, to cancel the command controller
- void **run ()**
- bool **last_command_timed_out ()**

5.13.1 Detailed Description

File: [command_controller.h](#) Desc: A [CommandController](#) manages the AutoCommands that make up an autonomous route. The AutoCommands are kept in a queue and get executed and removed from the queue in FIFO order.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 CommandController()

```
CommandController::CommandController (
    std::initializer_list< AutoCommand * > cmd ) [inline]
```

Create a [CommandController](#) with commands pre added. More can be added with [CommandController::add\(\)](#)

Parameters

<i>cmd</i>	
------------	--

5.13.3 Member Function Documentation

5.13.3.1 add() [1/3]

```
void CommandController::add (
    AutoCommand * cmd,
    double timeout_seconds = 10.0 )
```

File: [command_controller.cpp](#) Desc: A [CommandController](#) manages the AutoCommands that make up an autonomous route. The AutoCommands are kept in a queue and get executed and removed from the queue in FIFO order. Adds a command to the queue

Parameters

<i>cmd</i>	the AutoCommand we want to add to our list
<i>timeout_seconds</i> Generated by Doxygen	the number of seconds we will let the command run for. If it exceeds this, we cancel it and run on_timeout

5.13.3.2 add() [2/3]

```
void CommandController::add (
    std::vector< AutoCommand * > cmds )
```

Adds a command to the queue

Parameters

<i>cmd</i>	the AutoCommand we want to add to our list
<i>timeout_seconds</i>	the number of seconds we will let the command run for. If it exceeds this, we cancel it and run on_timeout. if it is ≤ 0 no time out will be applied

Add multiple commands to the queue. No timeout here.

Parameters

<i>cmds</i>	the AutoCommands we want to add to our list
-------------	---

5.13.3.3 add() [3/3]

```
void CommandController::add (
    std::vector< AutoCommand * > cmds,
    double timeout_sec )
```

Add multiple commands to the queue. No timeout here.

Parameters

<i>cmds</i>	the AutoCommands we want to add to our list Add multiple commands to the queue. No timeout here.
<i>cmds</i>	the AutoCommands we want to add to our list
<i>timeout_sec</i>	timeout in seconds to apply to all commands if they are still the default

Add multiple commands to the queue. No timeout here.

Parameters

<i>cmds</i>	the AutoCommands we want to add to our list
<i>timeout</i>	timeout in seconds to apply to all commands if they are still the default

5.13.3.4 add_cancel_func()

```
void CommandController::add_cancel_func (
    std::function< bool(void)> true_if_cancel )
```

`add_cancel_func` specifies that when this func evaluates to true, to cancel the command controller

Parameters

<i>true_if_cancel</i>	a function that returns true when we want to cancel the command controller
-----------------------	--

5.13.3.5 add_delay()

```
void CommandController::add_delay (
    int ms )
```

Adds a command that will delay progression of the queue

Parameters

<i>ms</i>	- number of milliseconds to wait before continuing execution of autonomous
-----------	--

5.13.3.6 last_command_timed_out()

```
bool CommandController::last_command_timed_out ( )
```

`last_command_timed_out` tells how the last command ended. Use this if you want to make decisions based on the end of the last command

Returns

true if the last command timed out. false if it finished regularly

5.13.3.7 run()

```
void CommandController::run ( )
```

Begin execution of the queue. Execute and remove commands in FIFO order

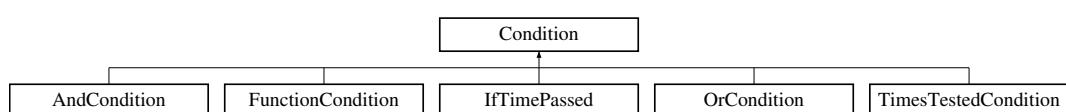
The documentation for this class was generated from the following files:

- include/utils/command_structure/command_controller.h
- src/utils/command_structure/command_controller.cpp

5.14 Condition Class Reference

```
#include <auto_command.h>
```

Inheritance diagram for Condition:



Public Member Functions

- `Condition * Or (Condition *b)`
- `Condition * And (Condition *b)`
- virtual bool `test ()=0`

5.14.1 Detailed Description

File: `auto_command.h` Desc: Interface for module-specific commands A `Condition` is a function that returns true or false `is_even` is a predicate that would return true if a number is even For our purposes, a `Condition` is a choice to be made at runtime `drive_sys.reached_point(10, 30)` is a predicate `time.has_elapsed(10, vex::seconds)` is a predicate extend this class for different choices you wish to make

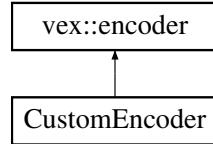
The documentation for this class was generated from the following files:

- `include/utils/command_structure/auto_command.h`
- `src/utils/command_structure/auto_command.cpp`

5.15 CustomEncoder Class Reference

```
#include <custom_encoder.h>
```

Inheritance diagram for CustomEncoder:



Public Member Functions

- `CustomEncoder (vex::triport::port &port, double ticks_per_rev)`
- void `setRotation (double val, vex::rotationUnits units)`
- void `setPosition (double val, vex::rotationUnits units)`
- double `rotation (vex::rotationUnits units)`
- double `position (vex::rotationUnits units)`
- double `velocity (vex::velocityUnits units)`

5.15.1 Detailed Description

A wrapper class for the vex encoder that allows the use of 3rd party encoders with different tick-per-revolution values.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 CustomEncoder()

```
CustomEncoder::CustomEncoder (
    vex::triport::port & port,
    double ticks_per_rev )
```

Construct an encoder with a custom number of ticks

Parameters

<i>port</i>	the triport port on the brain the encoder is plugged into
<i>ticks_per_rev</i>	the number of ticks the encoder will report for one revolution

5.15.3 Member Function Documentation

5.15.3.1 position()

```
double CustomEncoder::position (
    vex::rotationUnits units )
```

get the position that the encoder is at

Parameters

<i>units</i>	the unit we want the return value to be in
--------------	--

Returns

the position of the encoder in the units specified

5.15.3.2 rotation()

```
double CustomEncoder::rotation (
    vex::rotationUnits units )
```

get the rotation that the encoder is at

Parameters

<i>units</i>	the unit we want the return value to be in
--------------	--

Returns

the rotation of the encoder in the units specified

5.15.3.3 setPosition()

```
void CustomEncoder::setPosition (
    double val,
    vex::rotationUnits units )
```

sets the stored position of the encoder. Any further movements will be from this value

Parameters

<i>val</i>	the numerical value of the position we are setting to
<i>units</i>	the unit of val

5.15.3.4 setRotation()

```
void CustomEncoder::setRotation (
    double val,
    vex::rotationUnits units )
```

sets the stored rotation of the encoder. Any further movements will be from this value

Parameters

<i>val</i>	the numerical value of the angle we are setting to
<i>units</i>	the unit of val

5.15.3.5 velocity()

```
double CustomEncoder::velocity (
    vex::velocityUnits units )
```

get the velocity that the encoder is moving at

Parameters

<i>units</i>	the unit we want the return value to be in
--------------	--

Returns

the velocity of the encoder in the units specified

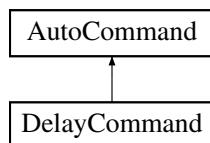
The documentation for this class was generated from the following files:

- include/subsystems/custom_encoder.h
- src/subsystems/custom_encoder.cpp

5.16 DelayCommand Class Reference

```
#include <delay_command.h>
```

Inheritance diagram for DelayCommand:



Public Member Functions

- [DelayCommand \(int ms\)](#)
- bool [run \(\) override](#)

Public Member Functions inherited from [AutoCommand](#)

- virtual void [on_timeout \(\)](#)
- [AutoCommand * withTimeout \(double t_seconds\)](#)
- [AutoCommand * withCancelCondition \(Condition *true_to_end\)](#)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- double [timeout_seconds](#) = default_timeout
- Condition * [true_to_end](#) = nullptr

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default_timeout](#) = 10.0

5.16.1 Detailed Description

File: [delay_command.h](#) Desc: A [DelayCommand](#) will make the robot wait the set amount of milliseconds before continuing execution of the autonomous route

5.16.2 Constructor & Destructor Documentation

5.16.2.1 [DelayCommand\(\)](#)

```
DelayCommand::DelayCommand (
    int ms ) [inline]
```

Construct a delay command

Parameters

<i>ms</i>	the number of milliseconds to delay for
-----------	---

5.16.3 Member Function Documentation

5.16.3.1 [run\(\)](#)

```
bool DelayCommand::run ( ) [inline], [override], [virtual]
```

Delays for the amount of milliseconds stored in the command Overrides run from [AutoCommand](#)

Returns

- true when complete

Reimplemented from [AutoCommand](#).

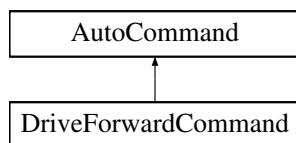
The documentation for this class was generated from the following file:

- include/utils/command_structure/delay_command.h

5.17 DriveForwardCommand Class Reference

```
#include <drive_commands.h>
```

Inheritance diagram for DriveForwardCommand:



Public Member Functions

- [DriveForwardCommand](#) ([TankDrive](#) &drive_sys, [Feedback](#) &feedback, double inches, directionType dir, double max_speed=1, double end_speed=0)
- bool [run](#) () override
- void [on_timeout](#) () override

Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand](#) * [withTimeout](#) (double t_seconds)
- [AutoCommand](#) * [withCancelCondition](#) ([Condition](#) *true_to_end)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- double [timeout_seconds](#) = default_timeout
- [Condition](#) * [true_to_end](#) = nullptr

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default_timeout](#) = 10.0

5.17.1 Detailed Description

[AutoCommand](#) wrapper class for the `drive_forward` function in the [TankDrive](#) class

5.17.2 Constructor & Destructor Documentation

5.17.2.1 DriveForwardCommand()

```
DriveForwardCommand::DriveForwardCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    double inches,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

File: [drive_commands.h](#) Desc: Holds all the [AutoCommand](#) subclasses that wrap (currently) [TankDrive](#) functions

Currently includes:

- `drive_forward`
- `turn_degrees`
- `drive_to_point`
- `turn_to_heading`
- `stop`

Also holds [AutoCommand](#) subclasses that wrap [OdometryBase](#) functions

Currently includes:

- `set_position` Construct a DriveForward Command

Parameters

<code>drive_sys</code>	the drive system we are commanding
<code>feedback</code>	the feedback controller we are using to execute the drive
<code>inches</code>	how far forward to drive
<code>dir</code>	the direction to drive
<code>max_speed</code>	0 -> 1 percentage of the drive systems speed to drive at

5.17.3 Member Function Documentation

5.17.3.1 on_timeout()

```
void DriveForwardCommand::on_timeout ( ) [override], [virtual]
```

Cleans up drive system if we time out before finishing

reset the drive system if we timeout

Reimplemented from [AutoCommand](#).

5.17.3.2 run()

```
bool DriveForwardCommand::run ( ) [override], [virtual]
```

Run drive_forward Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

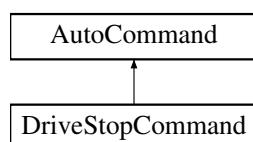
The documentation for this class was generated from the following files:

- include/utils/command_structure/drive_commands.h
- src/utils/command_structure/drive_commands.cpp

5.18 DriveStopCommand Class Reference

```
#include <drive_commands.h>
```

Inheritance diagram for DriveStopCommand:



Public Member Functions

- [DriveStopCommand \(TankDrive &drive_sys\)](#)
- [bool run \(\) override](#)
- [void on_timeout \(\) override](#)

Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand * withTimeout \(double t_seconds\)](#)
- [AutoCommand * withCancelCondition \(Condition *true_to_end\)](#)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- double `timeout_seconds` = `default_timeout`
- `Condition * true_to_end` = `nullptr`

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double `default_timeout` = 10.0

5.18.1 Detailed Description

[AutoCommand](#) wrapper class for the stop() function in the [TankDrive](#) class

5.18.2 Constructor & Destructor Documentation

5.18.2.1 `DriveStopCommand()`

```
DriveStopCommand::DriveStopCommand (   
    TankDrive & drive_sys )
```

Construct a DriveStop Command

Parameters

<code>drive_sys</code>	the drive system we are commanding
------------------------	------------------------------------

5.18.3 Member Function Documentation

5.18.3.1 `on_timeout()`

```
void DriveStopCommand::on_timeout ( ) [override], [virtual]
```

What to do if we timeout instead of finishing. timeout is specified by the timeout seconds in the constructor

Reimplemented from [AutoCommand](#).

5.18.3.2 `run()`

```
bool DriveStopCommand::run ( ) [override], [virtual]
```

Stop the drive system Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Stop the drive train Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

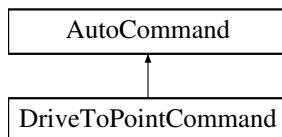
The documentation for this class was generated from the following files:

- include/utils/command_structure/drive_commands.h
- src/utils/command_structure/drive_commands.cpp

5.19 DriveToPointCommand Class Reference

```
#include <drive_commands.h>
```

Inheritance diagram for DriveToPointCommand:



Public Member Functions

- [DriveToPointCommand \(TankDrive &drive_sys, Feedback &feedback, double x, double y, directionType dir, double max_speed=1, double end_speed=0\)](#)
- [DriveToPointCommand \(TankDrive &drive_sys, Feedback &feedback, point_t point, directionType dir, double max_speed=1, double end_speed=0\)](#)
- bool [run \(\) override](#)

Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand * withTimeout \(double t_seconds\)](#)
- [AutoCommand * withCancelCondition \(Condition *true_to_end\)](#)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- double [timeout_seconds](#) = default_timeout
- [Condition * true_to_end](#) = nullptr

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default_timeout** = 10.0

5.19.1 Detailed Description

[AutoCommand](#) wrapper class for the drive_to_point function in the [TankDrive](#) class

5.19.2 Constructor & Destructor Documentation

5.19.2.1 DriveToPointCommand() [1/2]

```
DriveToPointCommand::DriveToPointCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    double x,
    double y,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

Construct a DriveForward Command

Parameters

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the drive
<i>x</i>	where to drive in the x dimension
<i>y</i>	where to drive in the y dimension
<i>dir</i>	the direction to drive
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

5.19.2.2 DriveToPointCommand() [2/2]

```
DriveToPointCommand::DriveToPointCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    point_t point,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

Construct a DriveForward Command

Parameters

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the drive
<i>point</i>	the point to drive to
<i>dir</i>	the direction to drive
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

5.19.3 Member Function Documentation

5.19.3.1 run()

```
bool DriveToPointCommand::run ( ) [override], [virtual]
```

Run drive_to_point Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command_structure/drive_commands.h
- src/utils/command_structure/drive_commands.cpp

5.20 AutoChooser::entry_t Struct Reference

```
#include <auto_chooser.h>
```

Public Attributes

- [Rect rect](#)
- std::string [name](#)

5.20.1 Detailed Description

[entry_t](#) is a datatype used to store information that the chooser knows about an auto selection button

5.20.2 Member Data Documentation

5.20.2.1 name

```
std::string AutoChooser::entry_t::name
```

name of the auto repretsented by the block

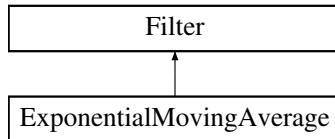
The documentation for this struct was generated from the following file:

- include/utils/auto_chooser.h

5.21 ExponentialMovingAverage Class Reference

```
#include <moving_average.h>
```

Inheritance diagram for ExponentialMovingAverage:



Public Member Functions

- [ExponentialMovingAverage \(int buffer_size\)](#)
- [ExponentialMovingAverage \(int buffer_size, double starting_value\)](#)
- void [add_entry \(double n\)](#) override
- double [get_value \(\) const](#) override
- int [get_size \(\)](#)

5.21.1 Detailed Description

[ExponentialMovingAverage](#)

An exponential moving average is a way of smoothing out noisy data. For many sensor readings, the noise is roughly symmetric around the actual value. This means that if you collect enough samples those that are too high are cancelled out by the samples that are too low leaving the real value.

A simple moving average lags significantly with time as it has to counteract old samples. An exponential moving average keeps more up to date by weighting newer readings higher than older readings so it is more up to date while also still smoothed.

The [ExponentialMovingAverage](#) class provides a simple interface to do this smoothing from our noisy sensor values.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 [ExponentialMovingAverage\(\) \[1/2\]](#)

```
ExponentialMovingAverage::ExponentialMovingAverage (
    int buffer_size )
```

Create a moving average calculator with 0 as the default value

Parameters

<i>buffer_size</i>	The size of the buffer. The number of samples that constitute a valid reading
--------------------	---

5.21.2.2 ExponentialMovingAverage() [2/2]

```
ExponentialMovingAverage::ExponentialMovingAverage (
    int buffer_size,
    double starting_value )
```

Create a moving average calculator with a specified default value

Parameters

<i>buffer_size</i>	The size of the buffer. The number of samples that constitute a valid reading
<i>starting_value</i>	The value that the average will be before any data is added

5.21.3 Member Function Documentation

5.21.3.1 add_entry()

```
void ExponentialMovingAverage::add_entry (
    double n ) [override], [virtual]
```

Add a reading to the buffer Before: [1 1 2 2 3 3] => 2 ^ After: [2 1 2 2 3 3] => 2.16 ^

Parameters

<i>n</i>	the sample that will be added to the moving average.
----------	--

Implements [Filter](#).

5.21.3.2 get_size()

```
int ExponentialMovingAverage::get_size ( )
```

How many samples the average is made from

Returns

the number of samples used to calculate this average

5.21.3.3 get_value()

```
double ExponentialMovingAverage::get_value ( ) const [override], [virtual]
```

Returns the average based off of all the samples collected so far

Returns

the calculated average. $\text{sum}(\text{samples})/\text{numsamples}$

How many samples the average is made from

Returns

the number of samples used to calculate this average

Implements [Filter](#).

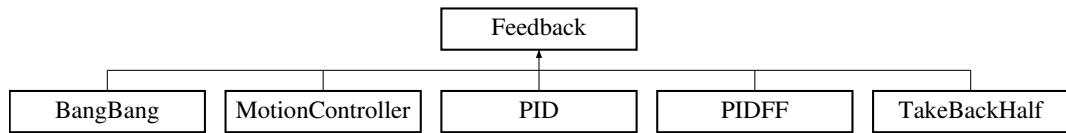
The documentation for this class was generated from the following files:

- include/utils/moving_average.h
- src/utils/moving_average.cpp

5.22 Feedback Class Reference

```
#include <feedback_base.h>
```

Inheritance diagram for Feedback:



Public Member Functions

- virtual void [init](#) (double start_pt, double set_pt, double start_vel=0.0, double end_vel=0.0)=0
- virtual double [update](#) (double val)=0
- virtual double [get](#) ()=0
- virtual void [set_limits](#) (double lower, double upper)=0
- virtual bool [is_on_target](#) ()=0

5.22.1 Detailed Description

Interface so that subsystems can easily switch between feedback loops

Author

Ryan McGee

Date

9/25/2022

5.22.2 Member Function Documentation

5.22.2.1 get()

```
virtual double Feedback::get ( ) [pure virtual]
```

Returns

the last saved result from the feedback controller

Implemented in [BangBang](#), [MotionController](#), [PID](#), [PIDFF](#), and [TakeBackHalf](#).

5.22.2.2 init()

```
virtual void Feedback::init (
    double start_pt,
    double set_pt,
    double start_vel = 0.0,
    double end_vel = 0.0 ) [pure virtual]
```

Initialize the feedback controller for a movement

Parameters

<i>start_pt</i>	the current sensor value
<i>set_pt</i>	where the sensor value should be
<i>start_vel</i>	Movement starting velocity
<i>end_vel</i>	Movement ending velocity

Implemented in [MotionController](#), [PIDFF](#), [PID](#), [BangBang](#), and [TakeBackHalf](#).

5.22.2.3 is_on_target()

```
virtual bool Feedback::is_on_target ( ) [pure virtual]
```

Returns

true if the feedback controller has reached it's setpoint

Implemented in [BangBang](#), [MotionController](#), [PID](#), [PIDFF](#), and [TakeBackHalf](#).

5.22.2.4 set_limits()

```
virtual void Feedback::set_limits (
    double lower,
    double upper ) [pure virtual]
```

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied.

Parameters

<i>lower</i>	Upper limit
<i>upper</i>	Lower limit

Implemented in [BangBang](#), [MotionController](#), [PID](#), [PIDFF](#), and [TakeBackHalf](#).

5.22.2.5 update()

```
virtual double Feedback::update (
    double val ) [pure virtual]
```

Iterate the feedback loop once with an updated sensor value

Parameters

<i>val</i>	value from the sensor
------------	-----------------------

Returns

feedback loop result

Implemented in [MotionController](#), [PID](#), [BangBang](#), [PIDFF](#), and [TakeBackHalf](#).

The documentation for this class was generated from the following file:

- include/utils/controls/feedback_base.h

5.23 FeedForward Class Reference

```
#include <feedforward.h>
```

Classes

- struct [ff_config_t](#)

Public Member Functions

- [FeedForward \(ff_config_t &cfg\)](#)
- double [calculate](#) (double v, double a, double pid_ref=0.0)
Perform the feedforward calculation.

5.23.1 Detailed Description

FeedForward

Stores the feedforward constants, and allows for quick computation. Feedforward should be used in systems that require smooth precise movements and have high inertia, such as drivetrains and lifts.

This is best used alongside a **PID** loop, with the form: `output = pid.get() + feedforward.calculate(v, a);`

In this case, the feedforward does the majority of the heavy lifting, and the pid loop only corrects for inconsistencies

For information about tuning feedforward, I recommend looking at this post: <https://www.chiefdelphi.com/t/paper-frc-drivetrain-characterization/160915> (yes I know it's for FRC but trust me, it's useful)

Author

Ryan McGee

Date

6/13/2022

5.23.2 Constructor & Destructor Documentation

5.23.2.1 FeedForward()

```
FeedForward::FeedForward (
    ff_config_t & cfg ) [inline]
```

Creates a **FeedForward** object.

Parameters

<code>cfg</code>	Configuration Struct for tuning
------------------	---------------------------------

5.23.3 Member Function Documentation

5.23.3.1 calculate()

```
double FeedForward::calculate (
    double v,
    double a,
    double pid_ref = 0.0 ) [inline]
```

Perform the feedforward calculation.

This calculation is the equation: $F = kG + kS * \text{sgn}(v) + kV * v + kA * a$

Parameters

<i>v</i>	Requested velocity of system
<i>a</i>	Requested acceleration of system

Returns

A feedforward that should closely represent the system if tuned correctly

The documentation for this class was generated from the following file:

- include/utils/controls/feedforward.h

5.24 FeedForward::ff_config_t Struct Reference

```
#include <feedforward.h>
```

Public Attributes

- double kS
- double kV
- double kA
- double kG

5.24.1 Detailed Description

`ff_config_t` holds the parameters to make the theoretical model of a real world system equation is of the form kS if the system is not stopped, 0 otherwise

- kV * desired velocity
- kA * desired acceleration
- kG

5.24.2 Member Data Documentation

5.24.2.1 kA

```
double FeedForward::ff_config_t::kA
```

kA - Acceleration coefficient: the power required to change the mechanism's speed. Multiplied by the requested acceleration.

5.24.2.2 kG

```
double FeedForward::ff_config_t::kG
```

kG - Gravity coefficient: only needed for lifts. The power required to overcome gravity and stay at steady state.

5.24.2.3 kS

```
double FeedForward::ff_config_t::kS
```

Coefficient to overcome static friction: the point at which the motor *starts* to move.

5.24.2.4 kV

```
double FeedForward::ff_config_t::kV
```

Velocity coefficient: the power required to keep the mechanism in motion. Multiplied by the requested velocity.

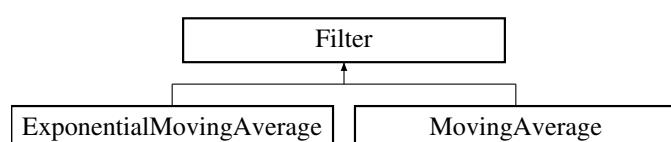
The documentation for this struct was generated from the following file:

- include/utils/controls/feedforward.h

5.25 Filter Class Reference

```
#include <moving_average.h>
```

Inheritance diagram for Filter:



Public Member Functions

- virtual void [add_entry](#) (double n)=0
- virtual double [get_value](#) () const =0

5.25.1 Detailed Description

Interface for filters Use add_entry to supply data and get_value to retrieve the filtered value

5.25.2 Member Function Documentation

5.25.2.1 add_entry()

```
virtual void Filter::add_entry (
    double n ) [pure virtual]
```

Implemented in [MovingAverage](#), and [ExponentialMovingAverage](#).

5.25.2.2 get_value()

```
virtual double Filter::get_value ( ) const [pure virtual]
```

Implemented in [MovingAverage](#), and [ExponentialMovingAverage](#).

The documentation for this class was generated from the following file:

- include/utils/moving_average.h

5.26 Flywheel Class Reference

```
#include <flywheel.h>
```

Public Member Functions

- [Flywheel](#) (vex::motor_group &motors, [Feedback](#) &feedback, [FeedForward](#) &helper, const double ratio, [Filter](#) &filt)
- double [get_target](#) () const
- double [getRPM](#) () const
- vex::motor_group & [get_motors](#) () const
- void [spin_manual](#) (double speed, directionType dir=fwd)
- void [spin_rpm](#) (double rpm)
- void [stop](#) ()
- bool [is_on_target](#) ()

check if the feedback controller thinks the flywheel is on target
- [screen::Page](#) * [Page](#) () const

Creates a page displaying info about the flywheel.
- [AutoCommand](#) * [SpinRpmCmd](#) (int rpm)

Creates a new auto command to spin the flywheel at the desired velocity.
- [AutoCommand](#) * [WaitUntilUpToSpeedCmd](#) ()

Creates a new auto command that will hold until the flywheel has its target as defined by its feedback controller.

Friends

- class [FlywheelPage](#)
- int [spinRPMTask](#) (void *wheelPointer)

5.26.1 Detailed Description

a [Flywheel](#) class that handles all control of a high inertia spinning disk. It gives multiple options for what control system to use in order to control wheel velocity and functions alerting the user when the flywheel is up to speed. [Flywheel](#) is a set and forget class. Once you create it you can call `spin_rpm` or `stop` on it at any time and it will take all necessary steps to accomplish this.

5.26.2 Constructor & Destructor Documentation

5.26.2.1 Flywheel()

```
Flywheel::Flywheel (
    vex::motor_group & motors,
    Feedback & feedback,
    FeedForward & helper,
    const double ratio,
    Filter & filt )
```

Create the [Flywheel](#) object using [PID](#) + feedforward for control.

Parameters

<i>motors</i>	pointer to the motors on the fly wheel
<i>feedback</i>	a feedback controller
<i>helper</i>	a feedforward config (only kV is used) to help the feedback controller along
<i>ratio</i>	ratio of the gears from the motor to the flywheel just multiplies the velocity
<i>filter</i>	the filter to use to smooth noisy motor readings

5.26.3 Member Function Documentation

5.26.3.1 get_motors()

```
motor_group & Flywheel::get_motors ( ) const
```

Returns the motors

Returns

the motors used to run the flywheel

5.26.3.2 get_target()

```
double Flywheel::get_target ( ) const
```

Return the target_rpm that the flywheel is currently trying to achieve

Returns

target_rpm the target rpm

Return the current value that the target_rpm should be set to

5.26.3.3 getRPM()

```
double Flywheel::getRPM ( ) const
```

return the velocity of the flywheel

5.26.3.4 is_on_target()

```
bool Flywheel::is_on_target ( ) [inline]
```

check if the feedback controller thinks the flywheel is on target

Returns

true if on target

5.26.3.5 Page()

```
screen::Page * Flywheel::Page ( ) const
```

Creates a page displaying info about the flywheel.

Returns

the page should be used for `screen::start_screen(screen, {fw.Page()});`

5.26.3.6 spin_manual()

```
void Flywheel::spin_manual (
    double speed,
    directionType dir = fwd )
```

Spin motors using voltage; defaults forward at 12 volts FOR USE BY OPCONTROL AND AUTONOMOUS - this only applies if the target_rpm thread is not running

Parameters

<i>speed</i>	- speed (between -1 and 1) to set the motor
<i>dir</i>	- direction that the motor moves in; defaults to forward

Spin motors using voltage; defaults forward at 12 volts FOR USE BY OPCONTROL AND AUTONOMOUS - this only applies if the RPM thread is not running

Parameters

<i>speed</i>	- speed (between -1 and 1) to set the motor
<i>dir</i>	- direction that the motor moves in; defaults to forward

5.26.3.7 spin_rpm()

```
void Flywheel::spin_rpm (
    double input_rpm )
```

starts or sets the target_rpm thread at new value what control scheme is dependent on control_style

Parameters

<i>rpm</i>	- the target_rpm we want to spin at
------------	-------------------------------------

starts or sets the RPM thread at new value what control scheme is dependent on control_style

Parameters

<i>input_rpm</i>	- set the current RPM
------------------	-----------------------

5.26.3.8 SpinRpmCmd()

```
AutoCommand * Flywheel::SpinRpmCmd (
    int rpm ) [inline]
```

Creates a new auto command to spin the flywheel at the desired velocity.

Parameters

<i>rpm</i>	the rpm to spin at
------------	--------------------

Returns

an auto command to add to a command controller

5.26.3.9 stop()

```
void Flywheel::stop ( )
```

Stops the motors. If manually spinning, this will do nothing just call spin_mainual(0.0) to send 0 volts

stop the RPM thread and the wheel

5.26.3.10 WaitUntilUpToSpeedCmd()

```
AutoCommand * Flywheel::WaitUntilUpToSpeedCmd ( ) [inline]
```

Creates a new auto command that will hold until the flywheel has its target as defined by its feedback controller.

Returns

an auto command to add to a command controller

5.26.4 Friends And Related Symbol Documentation

5.26.4.1 spinRPMTask

```
int spinRPMTask (
    void * wheelPointer ) [friend]
```

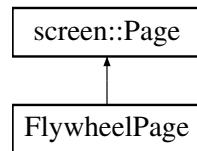
Runs a thread that keeps track of updating flywheel RPM and controlling it accordingly

The documentation for this class was generated from the following files:

- include/subsystems/flywheel.h
- src/subsystems/flywheel.cpp

5.27 FlywheelPage Class Reference

Inheritance diagram for FlywheelPage:



Public Member Functions

- **FlywheelPage** (const [Flywheel](#) &fw)
- void [update](#) (bool, int, int) override
- void [draw](#) (vex::brain::lcd &screen, bool, unsigned int) override

Static Public Attributes

- static const size_t [window_size](#) = 40

5.27.1 Member Function Documentation

5.27.1.1 draw()

```
void FlywheelPage::draw (
    vex::brain::lcd & screen,
    bool ,
    unsigned int ) [inline], [override], [virtual]
```

See also

[Page::draw](#)

Reimplemented from [screen::Page](#).

5.27.1.2 update()

```
void FlywheelPage::update (
    bool ,
    int ,
    int ) [inline], [override], [virtual]
```

See also

[Page::update](#)

Reimplemented from [screen::Page](#).

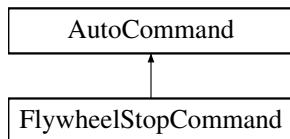
The documentation for this class was generated from the following file:

- [src/subsystems/flywheel.cpp](#)

5.28 FlywheelStopCommand Class Reference

```
#include <flywheel_commands.h>
```

Inheritance diagram for FlywheelStopCommand:



Public Member Functions

- [FlywheelStopCommand \(Flywheel &flywheel\)](#)
- bool [run \(\) override](#)

Public Member Functions inherited from [AutoCommand](#)

- virtual void [on_timeout \(\)](#)
- [AutoCommand * withTimeout \(double t_seconds\)](#)
- [AutoCommand * withCancelCondition \(Condition *true_to_end\)](#)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- double [timeout_seconds](#) = default_timeout
- [Condition * true_to_end](#) = nullptr

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default_timeout** = 10.0

5.28.1 Detailed Description

[AutoCommand](#) wrapper class for the stop function in the [Flywheel](#) class

5.28.2 Constructor & Destructor Documentation

5.28.2.1 FlywheelStopCommand()

```
FlywheelStopCommand::FlywheelStopCommand (
    Flywheel & flywheel )
```

Construct a [FlywheelStopCommand](#)

Parameters

<i>flywheel</i>	the flywheel system we are commanding
-----------------	---------------------------------------

5.28.3 Member Function Documentation

5.28.3.1 run()

```
bool FlywheelStopCommand::run ( ) [override], [virtual]
```

Run stop Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

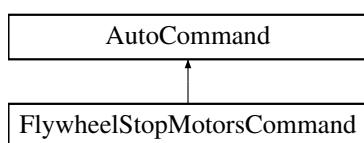
The documentation for this class was generated from the following files:

- include/utils/command_structure/flywheel_commands.h
- src/utils/command_structure/flywheel_commands.cpp

5.29 FlywheelStopMotorsCommand Class Reference

```
#include <flywheel_commands.h>
```

Inheritance diagram for FlywheelStopMotorsCommand:



Public Member Functions

- `FlywheelStopMotorsCommand (Flywheel &flywheel)`
- `bool run () override`

Public Member Functions inherited from `AutoCommand`

- `virtual void on_timeout ()`
- `AutoCommand * withTimeout (double t_seconds)`
- `AutoCommand * withCancelCondition (Condition *true_to_end)`

Additional Inherited Members

Public Attributes inherited from `AutoCommand`

- `double timeout_seconds = default_timeout`
- `Condition * true_to_end = nullptr`

Static Public Attributes inherited from `AutoCommand`

- `static constexpr double default_timeout = 10.0`

5.29.1 Detailed Description

`AutoCommand` wrapper class for the stopMotors function in the `Flywheel` class

5.29.2 Constructor & Destructor Documentation

5.29.2.1 `FlywheelStopMotorsCommand()`

```
FlywheelStopMotorsCommand::FlywheelStopMotorsCommand (
    Flywheel & flywheel )
```

Construct a FlywheelStopMotors Command

Parameters

<code>flywheel</code>	the flywheel system we are commanding
-----------------------	---------------------------------------

5.29.3 Member Function Documentation

5.29.3.1 `run()`

```
bool FlywheelStopMotorsCommand::run ( ) [override], [virtual]
```

Run stop Overrides run from `AutoCommand`

Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

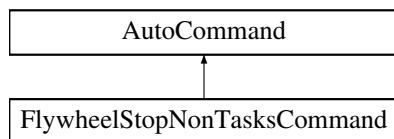
The documentation for this class was generated from the following files:

- include/utils/command_structure/flywheel_commands.h
- src/utils/command_structure/flywheel_commands.cpp

5.30 FlywheelStopNonTasksCommand Class Reference

```
#include <flywheel_commands.h>
```

Inheritance diagram for FlywheelStopNonTasksCommand:



Additional Inherited Members

Public Member Functions inherited from [AutoCommand](#)

- virtual void [on_timeout \(\)](#)
- [AutoCommand * withTimeout](#) (double t_seconds)
- [AutoCommand * withCancelCondition](#) ([Condition](#) *true_to_end)

Public Attributes inherited from [AutoCommand](#)

- double [timeout_seconds](#) = default_timeout
- [Condition](#) * [true_to_end](#) = nullptr

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default_timeout](#) = 10.0

5.30.1 Detailed Description

[AutoCommand](#) wrapper class for the stopNonTasks function in the [Flywheel](#) class

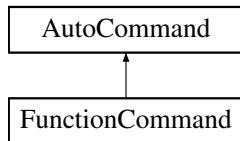
The documentation for this class was generated from the following files:

- include/utils/command_structure/flywheel_commands.h
- src/utils/command_structure/flywheel_commands.cpp

5.31 FunctionCommand Class Reference

```
#include <auto_command.h>
```

Inheritance diagram for FunctionCommand:



Public Member Functions

- **FunctionCommand** (std::function< bool(void)> f)
- bool **run** ()

Public Member Functions inherited from [AutoCommand](#)

- virtual void [on_timeout](#) ()
- [AutoCommand](#) * [withTimeout](#) (double t_seconds)
- [AutoCommand](#) * [withCancelCondition](#) ([Condition](#) *true_to_end)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- double [timeout_seconds](#) = default_timeout
- [Condition](#) * [true_to_end](#) = nullptr

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default_timeout](#) = 10.0

5.31.1 Detailed Description

[FunctionCommand](#) is fun and good way to do simple things Printing, launching nukes, and other quick and dirty one time things

5.31.2 Member Function Documentation

5.31.2.1 run()

```
bool FunctionCommand::run ( ) [inline], [virtual]
```

Executes the command Overridden by child classes

Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following file:

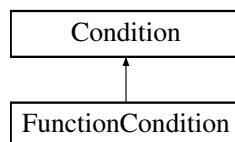
- include/utils/command_structure/auto_command.h

5.32 FunctionCondition Class Reference

[FunctionCondition](#) is a quick and dirty [Condition](#) to wrap some expression that should be evaluated at runtime.

```
#include <auto_command.h>
```

Inheritance diagram for FunctionCondition:



Public Member Functions

- [FunctionCondition](#) (std::function< bool()> cond, std::function< void(void)> timeout=[]() {})
- bool [test](#) () override

Public Member Functions inherited from [Condition](#)

- [Condition * Or](#) ([Condition](#) *b)
- [Condition * And](#) ([Condition](#) *b)

5.32.1 Detailed Description

[FunctionCondition](#) is a quick and dirty [Condition](#) to wrap some expression that should be evaluated at runtime.

5.32.2 Member Function Documentation

5.32.2.1 test()

```
bool FunctionCondition::test ( ) [override], [virtual]
```

Implements [Condition](#).

The documentation for this class was generated from the following files:

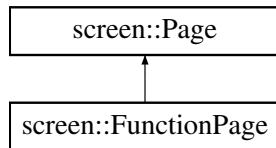
- include/utils/command_structure/auto_command.h
- src/utils/command_structure/auto_command.cpp

5.33 screen::FunctionPage Class Reference

Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything.

```
#include <screen.h>
```

Inheritance diagram for screen::FunctionPage:



Public Member Functions

- [FunctionPage](#) (update_func_t update_f, draw_func_t draw_t)
Creates a function page.
- void [update](#) (bool was_pressed, int x, int y) override
update uses the supplied update function to update this page
- void [draw](#) (vex::brain::lcd &, bool first_draw, unsigned int frame_number) override
draw uses the supplied draw function to draw to the screen

5.33.1 Detailed Description

Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 FunctionPage()

```
screen::FunctionPage::FunctionPage (
    update_func_t update_f,
    draw_func_t draw_f )
```

Creates a function page.

[FunctionPage](#).

Parameters

<i>update_f</i>	the function called every tick to respond to user input or do data collection
<i>draw_t</i>	the function called to draw to the screen
<i>update_f</i>	drawing function
<i>draw_f</i>	drawing function

5.33.3 Member Function Documentation**5.33.3.1 draw()**

```
void screen::FunctionPage::draw (
    vex::brain::lcd & screen,
    bool first_draw,
    unsigned int frame_number ) [override], [virtual]
```

draw uses the supplied draw function to draw to the screen

See also

[Page::draw](#)

Reimplemented from [screen::Page](#).

5.33.3.2 update()

```
void screen::FunctionPage::update (
    bool was_pressed,
    int x,
    int y ) [override], [virtual]
```

update uses the supplied update function to update this page

See also

[Page::update](#)

Reimplemented from [screen::Page](#).

The documentation for this class was generated from the following files:

- include/subsystems/screen.h
- src/subsystems/screen.cpp

5.34 GenericAuto Class Reference

```
#include <generic_auto.h>
```

Public Member Functions

- bool `run` (bool blocking)
- void `add` (state_ptr new_state)
- void `add_async` (state_ptr async_state)
- void `add_delay` (int ms)

5.34.1 Detailed Description

`GenericAuto` provides a pleasant interface for organizing an auto path steps of the path can be added with `add()` and when ready, calling `run()` will begin executing the path

5.34.2 Member Function Documentation

5.34.2.1 add()

```
void GenericAuto::add (
    state_ptr new_state )
```

Add a new state to the autonomous via function point of type "bool (ptr*)()"

Parameters

<code>new_state</code>	the function to run
------------------------	---------------------

5.34.2.2 add_async()

```
void GenericAuto::add_async (
    state_ptr async_state )
```

Add a new state to the autonomous via function point of type "bool (ptr*)()" that will run asynchronously

Parameters

<code>async_state</code>	the function to run
--------------------------	---------------------

5.34.2.3 add_delay()

```
void GenericAuto::add_delay (
    int ms )
```

`add_delay` adds a period where the auto system will simply wait for the specified time

Parameters

<code>ms</code>	how long to wait in milliseconds
-----------------	----------------------------------

5.34.2.4 run()

```
bool GenericAuto::run (
    bool blocking )
```

The method that runs the autonomous. If 'blocking' is true, then this method will run through every state until it finished.

If blocking is false, then assuming every state is also non-blocking, the method will run through the current state in the list and return immediately.

Parameters

<i>blocking</i>	Whether or not to block the thread until all states have run
-----------------	--

Returns

true after all states have finished.

The documentation for this class was generated from the following files:

- include/utils/generic_auto.h
- src/utils/generic_auto.cpp

5.35 GraphDrawer Class Reference

Public Member Functions

- [GraphDrawer](#) (int num_samples, double lower_bound, double upper_bound, std::vector< vex::color > colors, size_t num_series=1)

Creates a graph drawer with the specified number of series (each series is a separate line)
- void [add_samples](#) (std::vector< point_t > sample)
- void [add_samples](#) (std::vector< double > sample)
- void [draw](#) (vex::brain::lcd &screen, int x, int y, int width, int height)

5.35.1 Constructor & Destructor Documentation

5.35.1.1 GraphDrawer()

```
GraphDrawer::GraphDrawer (
    int num_samples,
    double lower_bound,
    double upper_bound,
    std::vector< vex::color > colors,
    size_t num_series = 1 )
```

Creates a graph drawer with the specified number of series (each series is a separate line)

Parameters

<i>num_samples</i>	the number of samples to graph at a time (40 will graph the last 40 data points)
<i>lower_bound</i>	the bottom of the window when displaying (if <i>upper_bound</i> = <i>lower_bound</i> , auto calculate bounds)
<i>upper_bound</i>	the top of the window when displaying (if <i>upper_bound</i> = <i>lower_bound</i> , auto calculate bounds)
<i>colors</i>	the colors of the series. must be of size <i>num_series</i>
<i>num_series</i>	the number of series to graph

5.35.2 Member Function Documentation**5.35.2.1 add_samples() [1/2]**

```
void GraphDrawer::add_samples (
    std::vector< double > sample )
```

add_samples adds a point to the graph, removing one from the back

Parameters

<i>sample</i>	a y coordinate of the next point to graph, the x coordinate is gotten from vex::timer::system(); (time in ms)
---------------	---

5.35.2.2 add_samples() [2/2]

```
void GraphDrawer::add_samples (
    std::vector< point_t > new_samples )
```

add_samples adds a point to the graph, removing one from the back

Parameters

<i>sample</i>	an x, y coordinate of the next point to graph
---------------	---

5.35.2.3 draw()

```
void GraphDrawer::draw (
    vex::brain::lcd & screen,
    int x,
    int y,
    int width,
    int height )
```

draws the graph to the screen in the constructor

Parameters

<i>x</i>	x position of the top left of the graphed region
----------	--

Parameters

<i>y</i>	y position of the top left of the graphed region
<i>width</i>	the width of the graphed region
<i>height</i>	the height of the graphed region

The documentation for this class was generated from the following files:

- include/utils/graph_drawer.h
- src/utils/graph_drawer.cpp

5.36 PurePursuit::hermite_point Struct Reference

```
#include <pure_pursuit.h>
```

Public Member Functions

- `point_t getPoint () const`
- `Vector2D getTangent () const`

Public Attributes

- double `x`
- double `y`
- double `dir`
- double `mag`

5.36.1 Detailed Description

a position along the hermite path contains a position and orientation information that the robot would be at at this point

The documentation for this struct was generated from the following file:

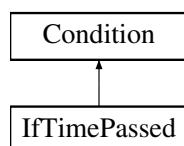
- include/utils/pure_pursuit.h

5.37 IfTimePassed Class Reference

`IfTimePassed` tests based on time since the command controller was constructed. Returns true if elapsed time > time_s.

```
#include <auto_command.h>
```

Inheritance diagram for IfTimePassed:



Public Member Functions

- **IfTimePassed** (double time_s)
- bool **test** () override

Public Member Functions inherited from [Condition](#)

- [Condition * Or](#) ([Condition *b](#))
- [Condition * And](#) ([Condition *b](#))

5.37.1 Detailed Description

[IfTimePassed](#) tests based on time since the command controller was constructed. Returns true if elapsed time > time_s.

5.37.2 Member Function Documentation

5.37.2.1 test()

```
bool IfTimePassed::test ( ) [override], [virtual]
```

Implements [Condition](#).

The documentation for this class was generated from the following files:

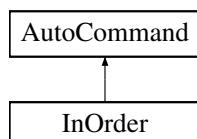
- include/utils/command_structure/auto_command.h
- src/utils/command_structure/auto_command.cpp

5.38 InOrder Class Reference

[InOrder](#) runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timeouts?

```
#include <auto_command.h>
```

Inheritance diagram for InOrder:



Public Member Functions

- **InOrder** (const [InOrder](#) &other)=default
- **InOrder** (std::queue<[AutoCommand](#) *> cmdqs)
- **InOrder** (std::initializer_list<[AutoCommand](#) *> cmdqs)
- bool **run** () override
- void **on_timeout** () override

Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand * withTimeout](#) (double t_seconds)
- [AutoCommand * withCancelCondition](#) ([Condition](#) *true_to_end)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- double [timeout_seconds](#) = default_timeout
- [Condition](#) * [true_to_end](#) = nullptr

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default_timeout](#) = 10.0

5.38.1 Detailed Description

[InOrder](#) runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timeouts?

[InOrder](#) runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timeouts?

5.38.2 Member Function Documentation

5.38.2.1 [on_timeout\(\)](#)

```
void InOrder::on_timeout ( ) [override], [virtual]
```

What to do if we timeout instead of finishing. timeout is specified by the timeout seconds in the constructor

Reimplemented from [AutoCommand](#).

5.38.2.2 [run\(\)](#)

```
bool InOrder::run ( ) [override], [virtual]
```

Executes the command Overridden by child classes

Returns

 true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- [include/utils/command_structure/auto_command.h](#)
- [src/utils/command_structure/auto_command.cpp](#)

5.39 screen::LabelConfig Struct Reference

Public Attributes

- std::string **label**

The documentation for this struct was generated from the following file:

- include/subsystems/screen.h

5.40 Lift< T > Class Template Reference

```
#include <lift.h>
```

Classes

- struct [lift_cfg_t](#)

Public Member Functions

- [Lift](#) (motor_group &lift_motors, [lift_cfg_t](#) &lift_cfg, map< T, double > &setpoint_map, limit *homing_switch=NULL)
- void [control_continuous](#) (bool up_ctrl, bool down_ctrl)
- void [control_manual](#) (bool up_btn, bool down_btn, int volt_up, int volt_down)
- void [control_setpoints](#) (bool up_step, bool down_step, vector< T > pos_list)
- bool [set_position](#) (T pos)
- bool [set_setpoint](#) (double val)
- double [get_setpoint](#) ()
- void [hold](#) ()
- void [home](#) ()
- bool [get_async](#) ()
- void [set_async](#) (bool val)
- void [set_sensor_function](#) (double(*fn_ptr)(void))
- void [set_sensor_reset](#) (void(*fn_ptr)(void))

5.40.1 Detailed Description

```
template<typename T>
class Lift< T >
```

LIFT A general class for lifts (e.g. 4bar, dr4bar, linear, etc) Uses a [PID](#) to hold the lift at a certain height under load, and to move the lift to different heights

Author

Ryan McGee

5.40.2 Constructor & Destructor Documentation

5.40.2.1 Lift()

```
template<typename T >
Lift< T >::Lift (
    motor_group & lift_motors,
    lift_cfg_t & lift_cfg,
    map< T, double > & setpoint_map,
    limit * homing_switch = NULL ) [inline]
```

Construct the `Lift` object and begin the background task that controls the lift.

Usage example: /code{.cpp} enum Positions {UP, MID, DOWN}; map<Positions, double> setpt_map { {DOWN, 0.0}, {MID, 0.5}, {UP, 1.0} }; Lift<Positions> my_lift(motors, lift_cfg, setpt_map); /endcode

Parameters

<code>lift_motors</code>	A set of motors, all set that positive rotation correlates with the lift going up
<code>lift_cfg</code>	<code>Lift</code> characterization information; PID tunings and movement speeds
<code>setpoint_map</code>	A map of enum type T, in which each enum entry corresponds to a different lift height

5.40.3 Member Function Documentation

5.40.3.1 control_continuous()

```
template<typename T >
void Lift< T >::control_continuous (
    bool up_ctrl,
    bool down_ctrl ) [inline]
```

Control the lift with an "up" button and a "down" button. Use `PID` to hold the lift when letting go.

Parameters

<code>up_ctrl</code>	Button controlling the "UP" motion
<code>down_ctrl</code>	Button controlling the "DOWN" motion

5.40.3.2 control_manual()

```
template<typename T >
void Lift< T >::control_manual (
    bool up_btn,
    bool down_btn,
    int volt_up,
    int volt_down ) [inline]
```

Control the lift with manual controls (no holding voltage)

Parameters

<i>up_btn</i>	Raise the lift when true
<i>down_btn</i>	Lower the lift when true
<i>volt_up</i>	Motor voltage when raising the lift
<i>volt_down</i>	Motor voltage when lowering the lift

5.40.3.3 control_setpoints()

```
template<typename T >
void Lift< T >::control_setpoints (
    bool up_step,
    bool down_step,
    vector< T > pos_list ) [inline]
```

Control the lift in "steps". When the "up" button is pressed, the lift will go to the next position as defined by pos_list.
Order matters!

Parameters

<i>up_step</i>	A button that increments the position of the lift.
<i>down_step</i>	A button that decrements the position of the lift.
<i>pos_list</i>	A list of positions for the lift to go through. The higher the index, the higher the lift should be (generally).

5.40.3.4 get_async()

```
template<typename T >
bool Lift< T >::get_async ( ) [inline]
```

Returns

whether or not the background thread is running the lift

5.40.3.5 get_setpoint()

```
template<typename T >
double Lift< T >::get_setpoint ( ) [inline]
```

Returns

The current setpoint for the lift

5.40.3.6 hold()

```
template<typename T >
void Lift< T >::hold ( ) [inline]
```

Target the class's setpoint. Calculate the PID output and set the lift motors accordingly.

5.40.3.7 home()

```
template<typename T >
void Lift< T >::home ( ) [inline]
```

A blocking function that automatically homes the lift based on a sensor or hard stop, and sets the position to 0. A watchdog times out after 3 seconds, to avoid damage.

5.40.3.8 set_async()

```
template<typename T >
void Lift< T >::set_async (
    bool val ) [inline]
```

Enables or disables the background task. Note that running the control functions, or set_position functions will immediately re-enable the task for autonomous use.

Parameters

<i>val</i>	Whether or not the background thread should run the lift
------------	--

5.40.3.9 set_position()

```
template<typename T >
bool Lift< T >::set_position (
    T pos ) [inline]
```

Enable the background task, and send the lift to a position, specified by the setpoint map from the constructor.

Parameters

<i>pos</i>	A lift position enum type
------------	---------------------------

Returns

True if the pid has reached the setpoint

5.40.3.10 set_sensor_function()

```
template<typename T >
void Lift< T >::set_sensor_function (
    double(*) (void) fn_ptr ) [inline]
```

Creates a custom hook for any other type of sensor to be used on the lift. Example: /code{.cpp} my_lift.set_sensor_function([](){return my_sensor.position();}); /endcode

Parameters

<i>fn_ptr</i>	Pointer to custom sensor function
---------------	-----------------------------------

5.40.3.11 set_sensor_reset()

```
template<typename T >
void Lift< T >::set_sensor_reset (
    void(*)(void) fn_ptr ) [inline]
```

Creates a custom hook to reset the sensor used in [set_sensor_function\(\)](#). Example: /code{.cpp} my_lift.set_sensor_reset(my_sensor.resetPosition); /endcode

5.40.3.12 set_setpoint()

```
template<typename T >
bool Lift< T >::set_setpoint (
    double val ) [inline]
```

Manually set a setpoint value for the lift [PID](#) to go to.

Parameters

<i>val</i>	Lift setpoint, in motor revolutions or sensor units defined by get_sensor . Cannot be outside the softstops.
------------	--

Returns

True if the pid has reached the setpoint

The documentation for this class was generated from the following file:

- include/subsystems/lift.h

5.41 Lift< T >::lift_cfg_t Struct Reference

```
#include <lift.h>
```

Public Attributes

- double **up_speed**
- double **down_speed**
- double **softstop_up**
- double **softstop_down**
- [PID::pid_config_t](#) **lift_pid_cfg**

5.41.1 Detailed Description

```
template<typename T>
struct Lift< T >::lift_cfg_t
```

[lift_cfg_t](#) holds the physical parameter specifications of a lify system. includes:

- maximum speeds for the system
- softstops to stop the lift from hitting the hard stops too hard

The documentation for this struct was generated from the following file:

- include/subsystems/lift.h

5.42 Logger Class Reference

Class to simplify writing to files.

```
#include <logger.h>
```

Public Member Functions

- **Logger** (const std::string &filename)
Create a logger that will save to a file.
- **Logger** (const **Logger** &l)=delete
copying not allowed
- **Logger** & **operator=** (const **Logger** &l)=delete
copying not allowed
- void **Log** (const std::string &s)
Write a string to the log.
- void **Log** (LogLevel level, const std::string &s)
Write a string to the log with a loglevel.
- void **LogIn** (const std::string &s)
Write a string and newline to the log.
- void **LogIn** (LogLevel level, const std::string &s)
Write a string and a newline to the log with a loglevel.
- void **Logf** (const char *fmt,...)
Write a formatted string to the log.
- void **Logf** (LogLevel level, const char *fmt,...)
Write a formatted string to the log with a loglevel.

Static Public Attributes

- static constexpr int **MAX_FORMAT_LEN** = 512
maximum size for a string to be before it's written

5.42.1 Detailed Description

Class to simplify writing to files.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 **Logger()**

```
Logger::Logger (
    const std::string & filename ) [explicit]
```

Create a logger that will save to a file.

Parameters

<i>filename</i>	the file to save to
-----------------	---------------------

5.42.3 Member Function Documentation

5.42.3.1 Log() [1/2]

```
void Logger::Log (
    const std::string & s )
```

Write a string to the log.

Parameters

<i>s</i>	the string to write
----------	---------------------

5.42.3.2 Log() [2/2]

```
void Logger::Log (
    LogLevel level,
    const std::string & s )
```

Write a string to the log with a loglevel.

Parameters

<i>level</i>	the level to write. DEBUG, NOTICE, WARNING, ERROR, CRITICAL, TIME
<i>s</i>	the string to write

5.42.3.3 Logf() [1/2]

```
void Logger::Logf (
    const char * fmt,
    ... )
```

Write a formatted string to the log.

Parameters

<i>fmt</i>	the format string (like printf)
...	the args

5.42.3.4 Logf() [2/2]

```
void Logger::Logf (
    LogLevel level,
    const char * fmt,
    ...
)
```

Write a formatted string to the log with a loglevel.

Parameters

<i>level</i>	the level to write. DEBUG, NOTICE, WARNING, ERROR, CRITICAL, TIME
<i>fmt</i>	the format string (like printf)
...	the args

5.42.3.5 Logln() [1/2]

```
void Logger::Logln (
    const std::string & s )
```

Write a string and newline to the log.

Parameters

<i>s</i>	the string to write
----------	---------------------

5.42.3.6 Logln() [2/2]

```
void Logger::Logln (
    LogLevel level,
    const std::string & s )
```

Write a string and a newline to the log with a loglevel.

Parameters

<i>level</i>	the level to write. DEBUG, NOTICE, WARNING, ERROR, CRITICAL, TIME
<i>s</i>	the string to write

The documentation for this class was generated from the following files:

- include/utils/logger.h
- src/utils/logger.cpp

5.43 MotionController::m_profile_cfg_t Struct Reference

```
#include <motion_controller.h>
```

Public Attributes

- double **max_v**
the maximum velocity the robot can drive
- double **accel**
the most acceleration the robot can do
- **PID::pid_config_t pid_cfg**
configuration parameters for the internal PID controller
- **FeedForward::ff_config_t ff_cfg**
configuration parameters for the internal

5.43.1 Detailed Description

m_profile_config holds all data the motion controller uses to plan paths. When motion profile is given a target to drive to, max_v and accel are used to make the trapezoid profile instructing the controller how to drive. pid_cfg, ff_cfg are used to find the motor outputs necessary to execute this path.

The documentation for this struct was generated from the following file:

- include/utils/controls/motion_controller.h

5.44 Mat2 Struct Reference

Public Member Functions

- **point_t operator* (const point_t p) const**

Static Public Member Functions

- static **Mat2 FromRotationDegrees (double degrees)**

Public Attributes

- double **X11**
- double **X12**
- double **X21**
- double **X22**

The documentation for this struct was generated from the following file:

- include/utils/geometry.h

5.45 StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage Class Reference

MaybeMessage a message of **Message** type or nothing. **MaybeMessage** m = {};// empty **MaybeMessage** m = **Message**::EnumField1.

```
#include <state_machine.h>
```

Public Member Functions

- **MaybeMessage ()**
Empty message - when theres no message.
- **MaybeMessage (Message msg)**
Create a maybe message with a message.
- **bool has_message ()**
check if the message is here
- **Message message ()**
Get the message stored. The return value is invalid unless has_message returned true.

5.45.1 Detailed Description

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log = false>
class StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage
```

MaybeMessage a message of Message type or nothing MaybeMessage m = {}; // empty MaybeMessage m = Message::EnumField1.

5.45.2 Constructor & Destructor Documentation

5.45.2.1 MaybeMessage()

```
template<typename System , typename IDType , typename Message , int32_t delay_ms, bool do_log
= false>
StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage::MaybeMessage (
    Message msg ) [inline]
```

Create a maybe message with a message.

Parameters

msg	the message to hold on to
-----	---------------------------

5.45.3 Member Function Documentation

5.45.3.1 has_message()

```
template<typename System , typename IDType , typename Message , int32_t delay_ms, bool do_log
= false>
bool StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage::has_message ( )
[inline]
```

check if the message is here

Returns

true if there is a message

5.45.3.2 message()

```
template<typename System , typename IDType , typename Message , int32_t delay_ms, bool do_log
= false>
Message StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage::message ( )
[inline]
```

Get the message stored. The return value is invalid unless has_message returned true.

Returns

The message if it exists. Undefined otherwise

The documentation for this class was generated from the following file:

- include/utils/state_machine.h

5.46 MecanumDrive Class Reference

```
#include <mecanum_drive.h>
```

Classes

- struct [mecanumdrive_config_t](#)

Public Member Functions

- [MecanumDrive](#) (vex::motor &left_front, vex::motor &right_front, vex::motor &left_rear, vex::motor &right_rear, vex::rotation *lateral_wheel=NULL, vex::inertial *imu=NULL, [mecanumdrive_config_t](#) *config=NULL)
- void [drive_raw](#) (double direction_deg, double magnitude, double rotation)
- void [drive](#) (double left_y, double left_x, double right_x, int power=2)
- bool [auto_drive](#) (double inches, double direction, double speed, bool gyro_correction=true)
- bool [auto_turn](#) (double degrees, double speed, bool ignore_imu=false)

5.46.1 Detailed Description

A class representing the Mecanum drivetrain. Contains 4 motors, a possible IMU (intertial), and a possible undriven perpendicular wheel.

5.46.2 Constructor & Destructor Documentation

5.46.2.1 MecanumDrive()

```
MecanumDrive::MecanumDrive (
    vex::motor & left_front,
    vex::motor & right_front,
    vex::motor & left_rear,
    vex::motor & right_rear,
    vex::rotation * lateral_wheel = NULL,
    vex::inertial * imu = NULL,
    mecanumdrive\_config\_t * config = NULL )
```

Create the Mecanum drivetrain object

5.46.3 Member Function Documentation

5.46.3.1 auto_drive()

```
bool MecanumDrive::auto_drive (
    double inches,
    double direction,
    double speed,
    bool gyro_correction = true )
```

Drive the robot in a straight line automatically. If the inertial was declared in the constructor, use it to correct while driving. If the lateral wheel was declared in the constructor, use it for more accurate positioning while strafing.

Parameters

<i>inches</i>	How far the robot should drive, in inches
<i>direction</i>	What direction the robot should travel in, in degrees. 0 is forward, +/-180 is reverse, clockwise is positive.
<i>speed</i>	The maximum speed the robot should travel, in percent: -1.0->+1.0
<i>gyro_correction</i>	=true Whether or not to use the gyro to help correct while driving. Will always be false if no gyro was declared in the constructor.

Drive the robot in a straight line automatically. If the inertial was declared in the constructor, use it to correct while driving. If the lateral wheel was declared in the constructor, use it for more accurate positioning while strafing.

Parameters

<i>inches</i>	How far the robot should drive, in inches
<i>direction</i>	What direction the robot should travel in, in degrees. 0 is forward, +/-180 is reverse, clockwise is positive.
<i>speed</i>	The maximum speed the robot should travel, in percent: -1.0->+1.0
<i>gyro_correction</i>	= true Whether or not to use the gyro to help correct while driving. Will always be false if no gyro was declared in the constructor.

Returns

Whether or not the maneuver is complete.

5.46.3.2 auto_turn()

```
bool MecanumDrive::auto_turn (
    double degrees,
    double speed,
    bool ignore_imu = false )
```

Autonomously turn the robot X degrees over it's center point. Uses a closed loop for control.

Parameters

<i>degrees</i>	How many degrees to rotate the robot. Clockwise positive.
<i>speed</i>	What percentage to run the motors at: 0.0 -> 1.0
<i>ignore_imu</i>	=false Whether or not to use the Inertial for determining angle. Will instead use circumference formula + robot's wheelbase + encoders to determine. Generated by Doxygen

Returns

whether or not the robot has finished the maneuver

Autonomously turn the robot X degrees over it's center point. Uses a closed loop for control.

Parameters

<i>degrees</i>	How many degrees to rotate the robot. Clockwise positive.
<i>speed</i>	What percentage to run the motors at: 0.0 -> 1.0
<i>ignore_imu</i>	= false Whether or not to use the Inertial for determining angle. Will instead use circumference formula + robot's wheelbase + encoders to determine.

Returns

whether or not the robot has finished the maneuver

5.46.3.3 drive()

```
void MecanumDrive::drive (
    double left_y,
    double left_x,
    double right_x,
    int power = 2 )
```

Drive the robot with a mecanum-style / arcade drive. Inputs are in percent (-100.0 -> 100.0) straight from the controller. Controls are mixed, so the robot can drive forward / strafe / rotate all at the same time.

Parameters

<i>left_y</i>	left joystick, Y axis (forward / backwards)
<i>left_x</i>	left joystick, X axis (strafe left / right)
<i>right_x</i>	right joystick, X axis (rotation left / right)
<i>power</i>	=2 how much of a "curve" there should be on drive controls; better for low speed maneuvers. Leave blank for a default curve of 2 (higher means more fidelity)

Drive the robot with a mecanum-style / arcade drive. Inputs are in percent (-100.0 -> 100.0) straight from the controller. Controls are mixed, so the robot can drive forward / strafe / rotate all at the same time.

Parameters

<i>left_y</i>	left joystick, Y axis (forward / backwards)
<i>left_x</i>	left joystick, X axis (strafe left / right)
<i>right_x</i>	right joystick, X axis (rotation left / right)
<i>power</i>	= 2 how much of a "curve" there should be on drive controls; better for low speed maneuvers. Leave blank for a default curve of 2 (higher means more fidelity)

5.46.3.4 drive_raw()

```
void MecanumDrive::drive_raw (
    double direction_deg,
    double magnitude,
    double rotation )
```

Drive the robot using vectors. This handles all the math required for mecanum control.

Parameters

<i>direction_deg</i>	the direction to drive the robot, in degrees. 0 is forward, 180 is back, clockwise is positive, counterclockwise is negative.
<i>magnitude</i>	How fast the robot should drive, in percent: 0.0->1.0
<i>rotation</i>	How fast the robot should rotate, in percent: -1.0->+1.0

The documentation for this class was generated from the following files:

- include/subsystems/mecanum_drive.h
- src/subsystems/mecanum_drive.cpp

5.47 MecanumDrive::mecanumdrive_config_t Struct Reference

```
#include <mecanum_drive.h>
```

Public Attributes

- [PID::pid_config_t drive_pid_conf](#)
- [PID::pid_config_t drive_gyro_pid_conf](#)
- [PID::pid_config_t turn_pid_conf](#)
- double **drive_wheel_diam**
- double **lateral_wheel_diam**
- double **wheelbase_width**

5.47.1 Detailed Description

Configure the Mecanum drive [PID](#) tunings and robot configurations

The documentation for this struct was generated from the following file:

- include/subsystems/mecanum_drive.h

5.48 motion_t Struct Reference

```
#include <trapezoid_profile.h>
```

Public Attributes

- **double pos**
1d position at this point in time
- **double vel**
1d velocity at this point in time
- **double accel**
1d acceleration at this point in time

5.48.1 Detailed Description

`motion_t` is a description of 1 dimensional motion at a point in time.

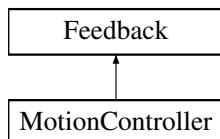
The documentation for this struct was generated from the following file:

- include/utils/controls/trapezoid_profile.h

5.49 MotionController Class Reference

```
#include <motion_controller.h>
```

Inheritance diagram for MotionController:



Classes

- struct `m_profile_cfg_t`

Public Member Functions

- `MotionController (m_profile_cfg_t &config)`
Construct a new Motion Controller object.
- `void init (double start_pt, double end_pt, double start_vel, double end_vel) override`
Initialize the motion profile for a new movement This will also reset the PID and profile timers.
- `double update (double sensor_val) override`
Update the motion profile with a new sensor value.
- `double get () override`
- `void set_limits (double lower, double upper) override`
- `bool is_on_target () override`
- `motion_t get_motion () const`
- `screen::Page * Page ()`

Static Public Member Functions

- static `FeedForward::ff_config_t tune_feedforward (TankDrive &drive, OdometryTank &odometry, double pct=0.6, double duration=2)`

Friends

- class **MotionControllerPage**

5.49.1 Detailed Description

Motion Controller class

This class defines a top-level motion profile, which can act as an intermediate between a subsystem class and the motors themselves

This takes the constants kS, kV, kA, kP, kI, kD, max_v and acceleration and wraps around a feedforward, **PID** and trapezoid profile. It does so with the following formula:

```
out = feedforward.calculate(motion_profile.get(time_s)) + pid.get(motion_profile.get(time_s))
```

For **PID** and Feedforward specific formulae, see [pid.h](#), [feedforward.h](#), and [trapezoid_profile.h](#)

Author

Ryan McGee

Date

7/13/2022

5.49.2 Constructor & Destructor Documentation

5.49.2.1 MotionController()

```
MotionController::MotionController (
    m_profile_cfg_t & config )
```

Construct a new Motion Controller object.

Parameters

<code>config</code>	The definition of how the robot is able to move max_v Maximum velocity the movement is capable of accel Acceleration / deceleration of the movement pid_cfg Definitions of kP, kI, and kD ff_cfg Definitions of kS, kV, and kA
---------------------	--

5.49.3 Member Function Documentation

5.49.3.1 get()

```
double MotionController::get ( ) [override], [virtual]
```

Returns

the last saved result from the feedback controller

Implements [Feedback](#).

5.49.3.2 get_motion()

```
motion_t MotionController::get_motion ( ) const
```

Returns

The current position, velocity and acceleration setpoints

5.49.3.3 init()

```
void MotionController::init (
    double start_pt,
    double end_pt,
    double start_vel,
    double end_vel ) [override], [virtual]
```

Initialize the motion profile for a new movement This will also reset the [PID](#) and profile timers.

Parameters

<i>start_pt</i>	Movement starting position
<i>end_pt</i>	Movement ending position
<i>start_vel</i>	Movement starting velocity
<i>end_vel</i>	Movement ending velocity

Implements [Feedback](#).

5.49.3.4 is_on_target()

```
bool MotionController::is_on_target ( ) [override], [virtual]
```

Returns

Whether or not the movement has finished, and the [PID](#) confirms it is on target

Implements [Feedback](#).

5.49.3.5 set_limits()

```
void MotionController::set_limits (
    double lower,
    double upper ) [override], [virtual]
```

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied. If limits are applied, the controller will not target any value below lower or above upper

Parameters

<i>lower</i>	upper limit
<i>upper</i>	lower limit

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied.

Parameters

<i>lower</i>	Upper limit
<i>upper</i>	Lower limit

Implements [Feedback](#).

5.49.3.6 tune_feedforward()

```
FeedForward::ff_config_t MotionController::tune_feedforward (
    TankDrive & drive,
    OdometryTank & odometry,
    double pct = 0.6,
    double duration = 2 ) [static]
```

This method attempts to characterize the robot's drivetrain and automatically tune the feedforward. It does this by first calculating the kS (voltage to overcome static friction) by slowly increasing the voltage until it moves.

Next is kV (voltage to sustain a certain velocity), where the robot will record its steady-state velocity at 'pct' speed.

Finally, kA (voltage needed to accelerate by a certain rate), where the robot will record the entire movement's velocity and acceleration, record a plot of [X=(pct-kV*V-kS), Y=(Acceleration)] along the movement, and since kA*Accel = pct-kV*V-kS, the reciprocal of the linear regression is the kA value.

Parameters

<i>drive</i>	The tankdrive to operate on
<i>odometry</i>	The robot's odometry subsystem
<i>pct</i>	Maximum velocity in percent (0->1.0)
<i>duration</i>	Amount of time the robot should be moving for the test

Returns

A tuned feedforward object

5.49.3.7 update()

```
double MotionController::update (
    double sensor_val )  [override], [virtual]
```

Update the motion profile with a new sensor value.

Parameters

<code>sensor_val</code>	Value from the sensor
-------------------------	-----------------------

Returns

the motor input generated from the motion profile

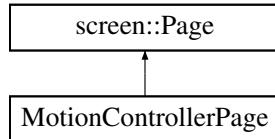
Implements [Feedback](#).

The documentation for this class was generated from the following files:

- include/utils/controls/motion_controller.h
- src/utils/controls/motion_controller.cpp

5.50 MotionControllerPage Class Reference

Inheritance diagram for MotionControllerPage:



Public Member Functions

- **MotionControllerPage** (const [MotionController](#) &mc)
- void [update](#) (bool was_pressed, int x, int y) override
collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this Page (only drawn page gets touch updates))
- void [draw](#) (vex::brain::lcd &screen, bool first_draw, unsigned int frame_number)
draw stored data to the screen (runs at 10 hz and only runs if this page is in front)

5.50.1 Member Function Documentation

5.50.1.1 draw()

```
void MotionControllerPage::draw (
    vex::brain::lcd & screen,
    bool first_draw,
    unsigned int frame_number )  [inline], [virtual]
```

draw stored data to the screen (runs at 10 hz and only runs if this page is in front)

Parameters

<code>first_draw</code>	true if we just switched to this page
<code>frame_number</code>	frame of drawing we are on (basically an animation tick)

Reimplemented from [screen::Page](#).

5.50.1.2 update()

```
void MotionControllerPage::update (
    bool was_pressed,
    int x,
    int y) [inline], [override], [virtual]
```

collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this Page (only drawn page gets touch updates))

Parameters

<code>was_pressed</code>	true if the screen has been pressed
<code>x</code>	x position of screen press (if the screen was pressed)
<code>y</code>	y position of screen press (if the screen was pressed)

Reimplemented from [screen::Page](#).

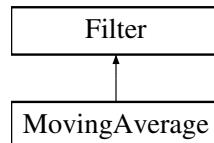
The documentation for this class was generated from the following file:

- [src/utils/controls/motion_controller.cpp](#)

5.51 MovingAverage Class Reference

```
#include <moving_average.h>
```

Inheritance diagram for MovingAverage:



Public Member Functions

- [MovingAverage](#) (int buffer_size)
- [MovingAverage](#) (int buffer_size, double starting_value)
- void [add_entry](#) (double n) override
- double [get_value](#) () const override
- int [get_size](#) () const

5.51.1 Detailed Description

MovingAverage

A moving average is a way of smoothing out noisy data. For many sensor readings, the noise is roughly symmetric around the actual value. This means that if you collect enough samples those that are too high are cancelled out by the samples that are too low leaving the real value.

The [MovingAverage](#) class provides a simple interface to do this smoothing from our noisy sensor values.

WARNING: because we need a lot of samples to get the actual value, the value given by the [MovingAverage](#) will 'lag' behind the actual value that the sensor is reading. Using a [MovingAverage](#) is thus a tradeoff between accuracy and lag time (more samples) vs. less accuracy and faster updating (less samples).

5.51.2 Constructor & Destructor Documentation

5.51.2.1 MovingAverage() [1/2]

```
MovingAverage::MovingAverage (
    int buffer_size )
```

Create a moving average calculator with 0 as the default value

Parameters

<i>buffer_size</i>	The size of the buffer. The number of samples that constitute a valid reading
--------------------	---

5.51.2.2 MovingAverage() [2/2]

```
MovingAverage::MovingAverage (
    int buffer_size,
    double starting_value )
```

Create a moving average calculator with a specified default value

Parameters

<i>buffer_size</i>	The size of the buffer. The number of samples that constitute a valid reading
<i>starting_value</i>	The value that the average will be before any data is added

5.51.3 Member Function Documentation

5.51.3.1 add_entry()

```
void MovingAverage::add_entry (
    double n ) [override], [virtual]
```

Add a reading to the buffer Before: [1 1 2 2 3 3] => 2 ^ After: [2 1 2 2 3 3] => 2.16 ^

Parameters

<code>n</code>	the sample that will be added to the moving average.
----------------	--

Implements [Filter](#).

5.51.3.2 `get_size()`

```
int MovingAverage::get_size ( ) const
```

How many samples the average is made from

Returns

the number of samples used to calculate this average

5.51.3.3 `get_value()`

```
double MovingAverage::get_value ( ) const [override], [virtual]
```

Returns the average based off of all the samples collected so far

Returns

the calculated average. sum(samples)/numsamples

How many samples the average is made from

Returns

the number of samples used to calculate this average

Implements [Filter](#).

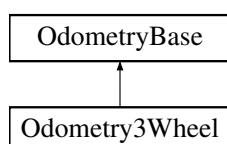
The documentation for this class was generated from the following files:

- include/utils/moving_average.h
- src/utils/moving_average.cpp

5.52 Odometry3Wheel Class Reference

```
#include <odometry_3wheel.h>
```

Inheritance diagram for Odometry3Wheel:



Classes

- struct `odometry3wheel_cfg_t`

Public Member Functions

- `Odometry3Wheel (CustomEncoder &lside_fwd, CustomEncoder &rside_fwd, CustomEncoder &off_axis, odometry3wheel_cfg_t &cfg, bool is_async=true)`
- `pose_t update () override`
- `void tune (vex::controller &con, TankDrive &drive)`

Public Member Functions inherited from `OdometryBase`

- `OdometryBase (bool is_async)`
- `pose_t get_position (void)`
- `virtual void set_position (const pose_t &newpos=zero_pos)`
- `AutoCommand * SetPositionCmd (const pose_t &newpos=zero_pos)`
- `void end_async ()`
- `double get_speed ()`
- `double get_accel ()`
- `double get_angular_speed_deg ()`
- `double get_angular_accel_deg ()`

Additional Inherited Members

Static Public Member Functions inherited from `OdometryBase`

- static int `background_task (void *ptr)`
- static double `pos_diff (pose_t start_pos, pose_t end_pos)`
- static double `rot_diff (pose_t pos1, pose_t pos2)`
- static double `smallest_angle (double start_deg, double end_deg)`

Public Attributes inherited from `OdometryBase`

- bool `end_task = false`
`end_task` is true if we instruct the odometry thread to shut down

Static Public Attributes inherited from `OdometryBase`

- static constexpr `pose_t zero_pos = {.x=0.0L, .y=0.0L, .rot=90.0L}`

Protected Attributes inherited from `OdometryBase`

- vex::task * `handle`
- vex::mutex `mut`
- `pose_t current_pos`
- `double speed`
- `double accel`
- `double ang_speed_deg`
- `double ang_accel_deg`

5.52.1 Detailed Description

Odometry3Wheel

This class handles the code for a standard 3-pod odometry setup, where there are 3 "pods" made up of undriven (dead) wheels connected to encoders in the following configuration:

```
+Y -----^ | | | | | | | O | | | | | | ==| | -----| +-----> + X
```

Where O is the center of rotation. The robot will monitor the changes in rotation of these wheels and calculate the robot's X, Y and rotation on the field.

This is a "set and forget" class, meaning once the object is created, the robot will immediately begin tracking its movement in the background.

Author

Ryan McGee

Date

Oct 31 2022

5.52.2 Constructor & Destructor Documentation

5.52.2.1 Odometry3Wheel()

```
Odometry3Wheel::Odometry3Wheel (
    CustomEncoder & lside_fwd,
    CustomEncoder & rside_fwd,
    CustomEncoder & off_axis,
    odometry3wheel_cfg_t & cfg,
    bool is_async = true )
```

Construct a new Odometry 3 Wheel object

Parameters

<i>lside_fwd</i>	left-side encoder reference
<i>rside_fwd</i>	right-side encoder reference
<i>off_axis</i>	off-axis (perpendicular) encoder reference
<i>cfg</i>	robot odometry configuration
<i>is_async</i>	true to constantly run in the background

5.52.3 Member Function Documentation

5.52.3.1 tune()

```
void Odometry3Wheel::tune (
    vex::controller & con,
    TankDrive & drive )
```

A guided tuning process to automatically find tuning parameters. This method is blocking, and returns when tuning has finished. Follow the instructions on the controller to complete the tuning process

Parameters

<i>con</i>	Controller reference, for screen and button control
<i>drive</i>	Drivetrain reference for robot control

A guided tuning process to automatically find tuning parameters. This method is blocking, and returns when tuning has finished. Follow the instructions on the controller to complete the tuning process

It is assumed the gear ratio and encoder PPR have been set correctly

5.52.3.2 update()

```
pose_t Odometry3Wheel::update ( ) [override], [virtual]
```

Update the current position of the robot once, using the current state of the encoders and the previous known location

Returns

the robot's updated position

Implements [OdometryBase](#).

The documentation for this class was generated from the following files:

- include/subsystems/odometry/odometry_3wheel.h
- src/subsystems/odometry/odometry_3wheel.cpp

5.53 Odometry3Wheel::odometry3wheel_cfg_t Struct Reference

```
#include <odometry_3wheel.h>
```

Public Attributes

- double *wheelbase_dist*
- double *off_axis_center_dist*
- double *wheel_diam*

5.53.1 Detailed Description

[odometry3wheel_cfg_t](#) holds all the specifications for how to calculate position with 3 encoders See the core wiki for what exactly each of these parameters measures

5.53.2 Member Data Documentation

5.53.2.1 off_axis_center_dist

```
double Odometry3Wheel::odometry3wheel_cfg_t::off_axis_center_dist
```

distance from the center of the robot to the center off axis wheel

5.53.2.2 wheel_diam

```
double Odometry3Wheel::odometry3wheel_cfg_t::wheel_diam
```

the diameter of the tracking wheel

5.53.2.3 wheelbase_dist

```
double Odometry3Wheel::odometry3wheel_cfg_t::wheelbase_dist
```

distance from the center of the left wheel to the center of the right wheel

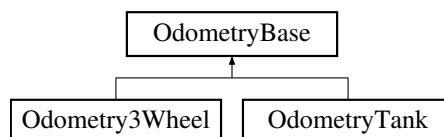
The documentation for this struct was generated from the following file:

- include/subsystems/odometry/odometry_3wheel.h

5.54 OdometryBase Class Reference

```
#include <odometry_base.h>
```

Inheritance diagram for OdometryBase:



Public Member Functions

- [OdometryBase \(bool is_async\)](#)
- [pose_t get_position \(void\)](#)
- virtual void [set_position \(const pose_t &newpos=zero_pos\)](#)
- [AutoCommand * SetPositionCmd \(const pose_t &newpos=zero_pos\)](#)
- virtual [pose_t update \(\)=0](#)
- void [end_async \(\)](#)
- double [get_speed \(\)](#)
- double [get_accel \(\)](#)
- double [get-angular_speed_deg \(\)](#)
- double [get-angular_accel_deg \(\)](#)

Static Public Member Functions

- static int `background_task` (void *ptr)
- static double `pos_diff` (`pose_t` start_pos, `pose_t` end_pos)
- static double `rot_diff` (`pose_t` pos1, `pose_t` pos2)
- static double `smallest_angle` (double start_deg, double end_deg)

Public Attributes

- bool `end_task` = false
end_task is true if we instruct the odometry thread to shut down

Static Public Attributes

- static constexpr `pose_t zero_pos` = {.x=0.0L, .y=0.0L, .rot=90.0L}

Protected Attributes

- vex::task * `handle`
- vex::mutex `mut`
- `pose_t current_pos`
- double `speed`
- double `accel`
- double `ang_speed_deg`
- double `ang_accel_deg`

5.54.1 Detailed Description

OdometryBase

This base class contains all the shared code between different implementations of odometry. It handles the asynchronous management, position input/output and basic math functions, and holds positional types specific to field orientation.

All future odometry implementations should extend this file and redefine `update()` function.

Author

Ryan McGee

Date

Aug 11 2021

5.54.2 Constructor & Destructor Documentation

5.54.2.1 OdometryBase()

```
OdometryBase::OdometryBase (
    bool is_async )
```

Construct a new Odometry Base object

Parameters

<i>is_async</i>	True to run constantly in the background, false to call update() manually
-----------------	---

5.54.3 Member Function Documentation

5.54.3.1 `background_task()`

```
int OdometryBase::background_task (
    void * ptr ) [static]
```

Function that runs in the background task. This function pointer is passed to the vex::task constructor.

Parameters

<i>ptr</i>	Pointer to OdometryBase object
------------	--

Returns

Required integer return code. Unused.

5.54.3.2 `end_async()`

```
void OdometryBase::end_async ( )
```

End the background task. Cannot be restarted. If the user wants to end the thread but keep the data up to date, they must run the [update\(\)](#) function manually from then on.

5.54.3.3 `get_accel()`

```
double OdometryBase::get_accel ( )
```

Get the current acceleration

Returns

the acceleration rate of the robot (inch/s²)

5.54.3.4 `get_angular_accel_deg()`

```
double OdometryBase::get_angular_accel_deg ( )
```

Get the current angular acceleration in degrees

Returns

the angular acceleration at which we are turning (deg/s²)

5.54.3.5 get_angular_speed_deg()

```
double OdometryBase::get_angular_speed_deg ( )
```

Get the current angular speed in degrees

Returns

the angular velocity at which we are turning (deg/s)

5.54.3.6 get_position()

```
pose_t OdometryBase::get_position (
    void )
```

Gets the current position and rotation

Returns

the position that the odometry believes the robot is at

Gets the current position and rotation

5.54.3.7 get_speed()

```
double OdometryBase::get_speed ( )
```

Get the current speed

Returns

the speed at which the robot is moving and grooving (inch/s)

5.54.3.8 pos_diff()

```
double OdometryBase::pos_diff (
    pose_t start_pos,
    pose_t end_pos ) [static]
```

Get the distance between two points

Parameters

<i>start_pos</i>	distance from this point
<i>end_pos</i>	to this point

Returns

the euclidean distance between start_pos and end_pos

5.54.3.9 rot_diff()

```
double OdometryBase::rot_diff (
    pose_t pos1,
    pose_t pos2 ) [static]
```

Get the change in rotation between two points

Parameters

<i>pos1</i>	position with initial rotation
<i>pos2</i>	position with final rotation

Returns

change in rotation between pos1 and pos2

Get the change in rotation between two points

5.54.3.10 set_position()

```
void OdometryBase::set_position (
    const pose_t & newpos = zero_pos ) [virtual]
```

Sets the current position of the robot

Parameters

<i>newpos</i>	the new position that the odometry will believe it is at
---------------	--

Sets the current position of the robot

Reimplemented in [OdometryTank](#).

5.54.3.11 smallest_angle()

```
double OdometryBase::smallest_angle (
    double start_deg,
    double end_deg ) [static]
```

Get the smallest difference in angle between a start heading and end heading. Returns the difference between -180 degrees and +180 degrees, representing the robot turning left or right, respectively.

Parameters

<code>start_deg</code>	initial angle (degrees)
<code>end_deg</code>	final angle (degrees)

Returns

the smallest angle from the initial to the final angle. This takes into account the wrapping of rotations around 360 degrees

Get the smallest difference in angle between a start heading and end heading. Returns the difference between -180 degrees and +180 degrees, representing the robot turning left or right, respectively.

5.54.3.12 update()

```
virtual pose_t OdometryBase::update ( ) [pure virtual]
```

Update the current position on the field based on the sensors

Returns

the location that the robot is at after the odometry does its calculations

Implemented in [Odometry3Wheel](#), and [OdometryTank](#).

5.54.4 Member Data Documentation**5.54.4.1 accel**

```
double OdometryBase::accel [protected]
```

the rate at which we are accelerating (inch/s²)

5.54.4.2 ang_accel_deg

```
double OdometryBase::ang_accel_deg [protected]
```

the rate at which we are accelerating our turn (deg/s²)

5.54.4.3 ang_speed_deg

```
double OdometryBase::ang_speed_deg [protected]
```

the speed at which we are turning (deg/s)

5.54.4.4 current_pos

`pose_t OdometryBase::current_pos [protected]`

Current position of the robot in terms of x,y,rotation

5.54.4.5 handle

`vex::task* OdometryBase::handle [protected]`

handle to the vex task that is running the odometry code

5.54.4.6 mut

`vex::mutex OdometryBase::mut [protected]`

Mutex to control multithreading

5.54.4.7 speed

`double OdometryBase::speed [protected]`

the speed at which we are travelling (inch/s)

5.54.4.8 zero_pos

`constexpr pose_t OdometryBase::zero_pos = { .x=0.0L, .y=0.0L, .rot=90.0L } [inline], [static], [constexpr]`

Zeroed position. X=0, Y=0, Rotation= 90 degrees

The documentation for this class was generated from the following files:

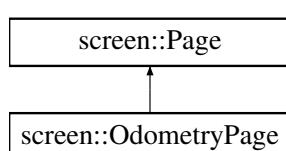
- `include/subsystems/odometry/odometry_base.h`
- `src/subsystems/odometry/odometry_base.cpp`

5.55 screen::OdometryPage Class Reference

a page that shows odometry position and rotation and a map (if an sd card with the file is on)

```
#include <screen.h>
```

Inheritance diagram for screen::OdometryPage:



Public Member Functions

- [OdometryPage \(OdometryBase &odom, double robot_width, double robot_height, bool do_trail\)](#)
Create an odometry trail. Make sure odometry is initialized before now.
- void [update \(bool was_pressed, int x, int y\) override](#)
- void [draw \(vex::brain::lcd &, bool first_draw, unsigned int frame_number\) override](#)

5.55.1 Detailed Description

a page that shows odometry position and rotation and a map (if an sd card with the file is on)

5.55.2 Constructor & Destructor Documentation

5.55.2.1 OdometryPage()

```
screen::OdometryPage::OdometryPage (
    OdometryBase & odom,
    double robot_width,
    double robot_height,
    bool do_trail )
```

Create an odometry trail. Make sure odometry is initialized before now.

Parameters

<i>odom</i>	the odometry system to monitor
<i>robot_width</i>	the width (side to side) of the robot in inches. Used for visualization
<i>robot_height</i>	the robot_height (front to back) of the robot in inches. Used for visualization
<i>do_trail</i>	whether or not to calculate and draw the trail. Drawing and storing takes a very <i>slight</i> extra amount of processing power

5.55.3 Member Function Documentation

5.55.3.1 draw()

```
void screen::OdometryPage::draw (
    vex::brain::lcd & scr,
    bool first_draw,
    unsigned int frame_number ) [override], [virtual]
```

See also

[Page::draw](#)

Reimplemented from [screen::Page](#).

5.55.3.2 update()

```
void screen::OdometryPage::update (
    bool was_pressed,
    int x,
    int y ) [override], [virtual]
```

See also

[Page::update](#)

Reimplemented from [screen::Page](#).

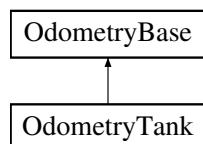
The documentation for this class was generated from the following files:

- [include/subsystems/screen.h](#)
- [src/subsystems/screen.cpp](#)

5.56 OdometryTank Class Reference

```
#include <odometry_tank.h>
```

Inheritance diagram for OdometryTank:



Public Member Functions

- [OdometryTank \(vex::motor_group &left_side, vex::motor_group &right_side, \[robot_specs_t\]\(#\) &config, vex::inertial *imu=NULL, bool is_async=true\)](#)
- [OdometryTank \(CustomEncoder &left_custom_enc, CustomEncoder &right_custom_enc, \[robot_specs_t\]\(#\) &config, vex::inertial *imu=NULL, bool is_async=true\)](#)
- [OdometryTank \(vex::encoder &left_vex_enc, vex::encoder &right_vex_enc, \[robot_specs_t\]\(#\) &config, vex::inertial *imu=NULL, bool is_async=true\)](#)
- [pose_t update \(\) override](#)
- [void set_position \(const pose_t &newpos=zero_pos\) override](#)

Public Member Functions inherited from [OdometryBase](#)

- [OdometryBase \(bool is_async\)](#)
- [pose_t get_position \(void\)](#)
- [AutoCommand * SetPositionCmd \(const pose_t &newpos=zero_pos\)](#)
- [void end_async \(\)](#)
- [double get_speed \(\)](#)
- [double get_accel \(\)](#)
- [double get_angular_speed_deg \(\)](#)
- [double get_angular_accel_deg \(\)](#)

Additional Inherited Members

Static Public Member Functions inherited from [OdometryBase](#)

- static int [background_task](#) (void *ptr)
- static double [pos_diff](#) ([pose_t](#) start_pos, [pose_t](#) end_pos)
- static double [rot_diff](#) ([pose_t](#) pos1, [pose_t](#) pos2)
- static double [smallest_angle](#) (double start_deg, double end_deg)

Public Attributes inherited from [OdometryBase](#)

- bool [end_task](#) = false
end_task is true if we instruct the odometry thread to shut down

Static Public Attributes inherited from [OdometryBase](#)

- static constexpr [pose_t zero_pos](#) = {.x=0.0L, .y=0.0L, .rot=90.0L}

Protected Attributes inherited from [OdometryBase](#)

- vex::task * [handle](#)
- vex::mutex [mut](#)
- [pose_t](#) [current_pos](#)
- double [speed](#)
- double [accel](#)
- double [ang_speed_deg](#)
- double [ang_accel_deg](#)

5.56.1 Detailed Description

[OdometryTank](#) defines an odometry system for a tank drivetrain. This requires encoders in the same orientation as the drive wheels. Odometry is a "start and forget" subsystem, which means once it's created and configured, it will constantly run in the background and track the robot's X, Y and rotation coordinates.

5.56.2 Constructor & Destructor Documentation

5.56.2.1 [OdometryTank\(\)](#) [1/3]

```
OdometryTank::OdometryTank (
    vex::motor_group & left_side,
    vex::motor_group & right_side,
    robot\_specs\_t & config,
    vex::inertial * imu = NULL,
    bool is_async = true )
```

Initialize the Odometry module, calculating position from the drive motors.

Parameters

<i>left_side</i>	The left motors
<i>right_side</i>	The right motors
<i>config</i>	the specifications that supply the odometry with descriptions of the robot. See robot_specs_t for what is contained
<i>imu</i>	The robot's inertial sensor. If not included, rotation is calculated from the encoders.
<i>is_async</i>	If true, position will be updated in the background continuously. If false, the programmer will have to manually call update() .

5.56.2.2 OdometryTank() [2/3]

```
OdometryTank::OdometryTank (
    CustomEncoder & left_custom_enc,
    CustomEncoder & right_custom_enc,
    robot_specs_t & config,
    vex::inertial * imu = NULL,
    bool is_async = true )
```

Initialize the Odometry module, calculating position from the drive motors.

Parameters

<i>left_custom_enc</i>	The left custom encoder
<i>right_custom_enc</i>	The right custom encoder
<i>config</i>	the specifications that supply the odometry with descriptions of the robot. See robot_specs_t for what is contained
<i>imu</i>	The robot's inertial sensor. If not included, rotation is calculated from the encoders.
<i>is_async</i>	If true, position will be updated in the background continuously. If false, the programmer will have to manually call update() .

5.56.2.3 OdometryTank() [3/3]

```
OdometryTank::OdometryTank (
    vex::encoder & left_vex_enc,
    vex::encoder & right_vex_enc,
    robot_specs_t & config,
    vex::inertial * imu = NULL,
    bool is_async = true )
```

Initialize the Odometry module, calculating position from the drive motors.

Parameters

<i>left_vex_enc</i>	The left vex encoder
<i>right_vex_enc</i>	The right vex encoder
<i>config</i>	the specifications that supply the odometry with descriptions of the robot. See robot_specs_t for what is contained
<i>imu</i>	The robot's inertial sensor. If not included, rotation is calculated from the encoders.
<i>is_async</i>	If true, position will be updated in the background continuously. If false, the programmer will have to manually call update() .

5.56.3 Member Function Documentation

5.56.3.1 set_position()

```
void OdometryTank::set_position (
    const pose_t & newpos = zero_pos ) [override], [virtual]
```

set_position tells the odometry to place itself at a position

Parameters

<code>newpos</code>	the position the odometry will take
---------------------	-------------------------------------

Resets the position and rotational data to the input.

Reimplemented from [OdometryBase](#).

5.56.3.2 update()

```
pose_t OdometryTank::update () [override], [virtual]
```

Update the current position on the field based on the sensors

Returns

the position that odometry has calculated itself to be at

Update, store and return the current position of the robot. Only use if not initializing with a separate thread.

Implements [OdometryBase](#).

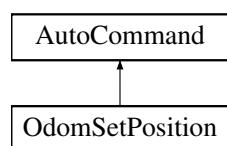
The documentation for this class was generated from the following files:

- include/subsystems/odometry/odometry_tank.h
- src/subsystems/odometry/odometry_tank.cpp

5.57 OdomSetPosition Class Reference

```
#include <drive_commands.h>
```

Inheritance diagram for OdomSetPosition:



Public Member Functions

- `OdomSetPosition (OdometryBase &odom, const pose_t &newpos=OdometryBase::zero_pos)`
- `bool run () override`

Public Member Functions inherited from [AutoCommand](#)

- `virtual void on_timeout ()`
- `AutoCommand * withTimeout (double t_seconds)`
- `AutoCommand * withCancelCondition (Condition *true_to_end)`

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- `double timeout_seconds = default_timeout`
- `Condition * true_to_end = nullptr`

Static Public Attributes inherited from [AutoCommand](#)

- `static constexpr double default_timeout = 10.0`

5.57.1 Detailed Description

[AutoCommand](#) wrapper class for the set_position function in the Odometry class

5.57.2 Constructor & Destructor Documentation

5.57.2.1 OdomSetPosition()

```
OdomSetPosition::OdomSetPosition (
    OdometryBase & odom,
    const pose_t & newpos = OdometryBase::zero_pos )
```

constructs a new [OdomSetPosition](#) command

Parameters

<code>odom</code>	the odometry system we are setting
<code>newpos</code>	the position we are telling the odometry to take. defaults to (0, 0), angle = 90

Construct an Odometry set pos

Parameters

<code>odom</code>	the odometry system we are setting
<code>newpos</code>	the now position to set the odometry to

5.57.3 Member Function Documentation

5.57.3.1 run()

```
bool OdomSetPosition::run ( ) [override], [virtual]
```

Run set_position Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

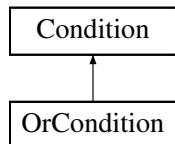
Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command_structure/drive_commands.h
- src/utils/command_structure/drive_commands.cpp

5.58 OrCondition Class Reference

Inheritance diagram for OrCondition:



Public Member Functions

- [OrCondition \(Condition *A, Condition *B\)](#)
- bool [test \(\) override](#)

Public Member Functions inherited from [Condition](#)

- [Condition * Or \(Condition *b\)](#)
- [Condition * And \(Condition *b\)](#)

5.58.1 Member Function Documentation

5.58.1.1 test()

```
bool OrCondition::test ( ) [inline], [override], [virtual]
```

Implements [Condition](#).

The documentation for this class was generated from the following file:

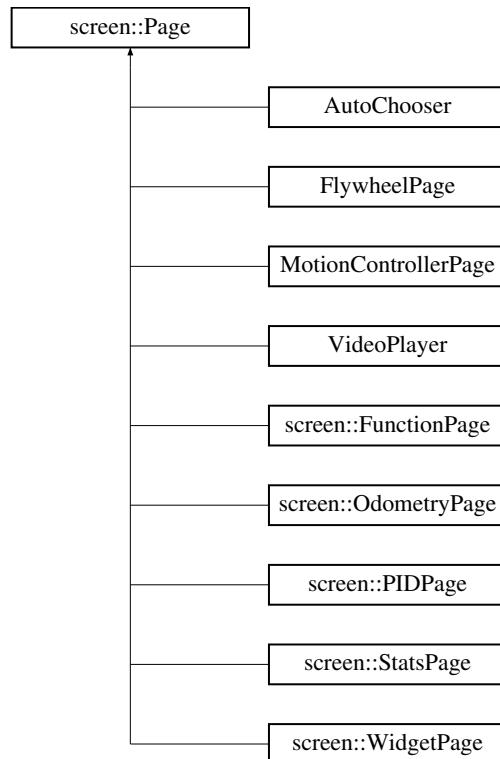
- src/utils/command_structure/auto_command.cpp

5.59 screen::Page Class Reference

[Page](#) describes one part of the screen slideshow.

```
#include <screen.h>
```

Inheritance diagram for screen::Page:



Public Member Functions

- virtual void [update](#) (bool was_pressed, int x, int y)
collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this [Page](#) (only drawn page gets touch updates))
- virtual void [draw](#) (vex::brain::lcd &screen, bool first_draw, unsigned int frame_number)
draw stored data to the screen (runs at 10 hz and only runs if this page is in front)

5.59.1 Detailed Description

[Page](#) describes one part of the screen slideshow.

5.59.2 Member Function Documentation

5.59.2.1 draw()

```
virtual void screen::Page::draw (
    vex::brain::lcd & screen,
    bool first_draw,
    unsigned int frame_number ) [virtual]
```

draw stored data to the screen (runs at 10 hz and only runs if this page is in front)

Parameters

<i>first_draw</i>	true if we just switched to this page
<i>frame_number</i>	frame of drawing we are on (basically an animation tick)

Reimplemented in [AutoChooser](#), [screen::WidgetPage](#), [screen::StatsPage](#), [screen::OdometryPage](#), [screen::FunctionPage](#), [screen::PIDPage](#), [MotionControllerPage](#), [VideoPlayer](#), and [FlywheelPage](#).

5.59.2.2 update()

```
virtual void screen::Page::update (
    bool was_pressed,
    int x,
    int y) [virtual]
```

collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this [Page](#) (only drawn page gets touch updates))

Parameters

<i>was_pressed</i>	true if the screen has been pressed
<i>x</i>	x position of screen press (if the screen was pressed)
<i>y</i>	y position of screen press (if the screen was pressed)

Reimplemented in [AutoChooser](#), [VideoPlayer](#), [screen::WidgetPage](#), [screen::StatsPage](#), [screen::OdometryPage](#), [screen::FunctionPage](#), [screen::PIDPage](#), [MotionControllerPage](#), and [FlywheelPage](#).

The documentation for this class was generated from the following file:

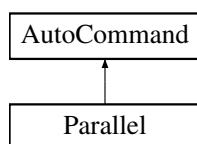
- include/subsystems/screen.h

5.60 Parallel Class Reference

[Parallel](#) runs multiple commands in parallel and waits for all to finish before continuing. if none finish before this command's timeout, it will call `on_timeout` on all children continue.

```
#include <auto_command.h>
```

Inheritance diagram for Parallel:



Public Member Functions

- `Parallel (std::initializer_list< AutoCommand * > cmd)`
- `bool run () override`
- `void on_timeout () override`

Public Member Functions inherited from [AutoCommand](#)

- `AutoCommand * withTimeout (double t_seconds)`
- `AutoCommand * withCancelCondition (Condition *true_to_end)`

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- `double timeout_seconds = default_timeout`
- `Condition * true_to_end = nullptr`

Static Public Attributes inherited from [AutoCommand](#)

- `static constexpr double default_timeout = 10.0`

5.60.1 Detailed Description

`Parallel` runs multiple commands in parallel and waits for all to finish before continuing. if none finish before this command's timeout, it will call `on_timeout` on all children continue.

5.60.2 Member Function Documentation

5.60.2.1 `on_timeout()`

```
void Parallel::on_timeout ( ) [override], [virtual]
```

What to do if we timeout instead of finishing. timeout is specified by the timeout seconds in the constructor

Reimplemented from [AutoCommand](#).

5.60.2.2 `run()`

```
bool Parallel::run ( ) [override], [virtual]
```

Executes the command Overridden by child classes

Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- `include/utils/command_structure/auto_command.h`
- `src/utils/command_structure/auto_command.cpp`

5.61 parallel_runner_info Struct Reference

Public Attributes

- int **index**
- std::vector< vex::task * > * **runners**
- AutoCommand * **cmd**

The documentation for this struct was generated from the following file:

- src/utils/command_structure/auto_command.cpp

5.62 PurePursuit::Path Class Reference

```
#include <pure_pursuit.h>
```

Public Member Functions

- Path (std::vector< point_t > points, double radius)
- std::vector< point_t > get_points ()
- double get_radius ()
- bool is_valid ()

5.62.1 Detailed Description

Wrapper for a vector of points, checking if any of the points are too close for pure pursuit

5.62.2 Constructor & Destructor Documentation

5.62.2.1 Path()

```
PurePursuit::Path::Path (
    std::vector< point_t > points,
    double radius )
```

Create a [Path](#)

Parameters

<i>points</i>	the points that make up the path
<i>radius</i>	the lookahead radius for pure pursuit

5.62.3 Member Function Documentation

5.62.3.1 get_points()

```
std::vector< point_t > PurePursuit::Path::get_points ( )
```

Get the points associated with this Path

5.62.3.2 get_radius()

```
double PurePursuit::Path::get_radius ( )
```

Get the radius associated with this Path

5.62.3.3 is_valid()

```
bool PurePursuit::Path::is_valid ( )
```

Get whether this path will behave as expected

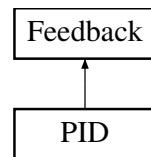
The documentation for this class was generated from the following files:

- include/utils/pure_pursuit.h
- src/utils/pure_pursuit.cpp

5.63 PID Class Reference

```
#include <pid.h>
```

Inheritance diagram for PID:



Classes

- struct [pid_config_t](#)

Public Types

- enum [ERROR_TYPE](#) { [LINEAR](#) , [ANGULAR](#) }

Public Member Functions

- `PID (pid_config_t &config)`
- `void init (double start_pt, double set_pt, double start_vel=0, double end_vel=0) override`
- `double update (double sensor_val) override`
- `double get_sensor_val () const`
`gets the sensor value that we were last updated with`
- `double get () override`
- `void set_limits (double lower, double upper) override`
- `bool is_on_target () override`
- `void reset ()`
- `double get_error ()`
- `double get_target () const`
- `void set_target (double target)`

Public Attributes

- `pid_config_t & config`

5.63.1 Detailed Description

PID Class

Defines a standard feedback loop using the constants kP, kI, kD, deadband, and on_target_time. The formula is:

`out = kP*error + kI*integral(d Error) + kD*(dError/dt)`

The `PID` object will determine it is "on target" when the error is within the deadband, for a duration of `on_target_time`

Author

Ryan McGee

Date

4/3/2020

5.63.2 Member Enumeration Documentation

5.63.2.1 ERROR_TYPE

```
enum PID::ERROR_TYPE
```

An enum to distinguish between a linear and angular calculation of `PID` error.

5.63.3 Constructor & Destructor Documentation

5.63.3.1 PID()

```
PID::PID (
    pid_config_t & config )
```

Create the `PID` object

Parameters

<code>config</code>	the configuration data for this controller
---------------------	--

Create the [PID](#) object

5.63.4 Member Function Documentation

5.63.4.1 `get()`

```
double PID::get ( ) [override], [virtual]
```

Gets the current [PID](#) out value, from when [update\(\)](#) was last run

Returns

the Out value of the controller (voltage, RPM, whatever the [PID](#) controller is controlling)

Gets the current [PID](#) out value, from when [update\(\)](#) was last run

Implements [Feedback](#).

5.63.4.2 `get_error()`

```
double PID::get_error ( )
```

Get the delta between the current sensor data and the target

Returns

the error calculated. how it is calculated depends on error_method specified in [pid_config_t](#)

Get the delta between the current sensor data and the target

5.63.4.3 `get_sensor_val()`

```
double PID::get_sensor_val ( ) const
```

gets the sensor value that we were last updated with

Returns

`sensor_val`

5.63.4.4 get_target()

```
double PID::get_target ( ) const
```

Get the [PID](#)'s target

Returns

the target the [PID](#) controller is trying to achieve

5.63.4.5 init()

```
void PID::init (
    double start_pt,
    double set_pt,
    double start_vel = 0,
    double end_vel = 0 ) [override], [virtual]
```

Inherited from [Feedback](#) for interoperability. Update the setpoint and reset integral accumulation

`start_pt` can be safely ignored in this feedback controller

Parameters

<code>start_pt</code>	completely ignored for PID . necessary to satisfy Feedback base
<code>set_pt</code>	sets the target of the PID controller
<code>start_vel</code>	completely ignored for PID . necessary to satisfy Feedback base
<code>end_vel</code>	sets the target end velocity of the PID controller

Implements [Feedback](#).

5.63.4.6 is_on_target()

```
bool PID::is_on_target ( ) [override], [virtual]
```

Checks if the [PID](#) controller is on target.

Returns

true if the loop is within [deadband] for [on_target_time] seconds

Returns true if the loop is within [deadband] for [on_target_time] seconds

Implements [Feedback](#).

5.63.4.7 reset()

```
void PID::reset ( )
```

Reset the [PID](#) loop by resetting time since 0 and accumulated error.

5.63.4.8 set_limits()

```
void PID::set_limits (
    double lower,
    double upper ) [override], [virtual]
```

Set the limits on the [PID](#) out. The [PID](#) out will "clip" itself to be between the limits.

Parameters

<i>lower</i>	the lower limit. the PID controller will never command the output go below <i>lower</i>
<i>upper</i>	the upper limit. the PID controller will never command the output go higher than <i>upper</i>

Set the limits on the [PID](#) out. The [PID](#) out will "clip" itself to be between the limits.

Implements [Feedback](#).

5.63.4.9 set_target()

```
void PID::set_target (
    double target )
```

Set the target for the [PID](#) loop, where the robot is trying to end up

Parameters

<i>target</i>	the sensor reading we would like to achieve
---------------	---

Set the target for the [PID](#) loop, where the robot is trying to end up

5.63.4.10 update()

```
double PID::update (
    double sensor_val ) [override], [virtual]
```

Update the [PID](#) loop by taking the time difference from last update, and running the [PID](#) formula with the new sensor data

Parameters

<i>sensor_val</i>	the distance, angle, encoder position or whatever it is we are measuring
-------------------	--

Returns

the new output. What would be returned by [PID::get\(\)](#)

Implements [Feedback](#).

5.63.5 Member Data Documentation

5.63.5.1 config

`pid_config_t& PID::config`

configuration struct for this controller. see [pid_config_t](#) for information about what this contains

The documentation for this class was generated from the following files:

- include/utils/controls/pid.h
- src/utils/controls/pid.cpp

5.64 PID::pid_config_t Struct Reference

#include <pid.h>

Public Attributes

- double **p**
*proportional coefficient p * error()*
- double **i**
*integral coefficient i * integral(error)*
- double **d**
*derivative coefficient d * derivative(error)*
- double **deadband**
at what threshold are we close enough to be finished
- double **on_target_time**
- [ERROR_TYPE](#) **error_method**

5.64.1 Detailed Description

`pid_config_t` holds the configuration parameters for a pid controller In addition to the constant of proportional, integral and derivative, these parameters include:

- deadband -
- on_target_time - for how long do we have to be at the target to stop As well, `pid_config_t` holds an error type which determines whether errors should be calculated as if the sensor position is a measure of distance or an angle

5.64.2 Member Data Documentation

5.64.2.1 error_method

`ERROR_TYPE PID::pid_config_t::error_method`

Linear or angular. wheter to do error as a simple subtraction or to wrap

5.64.2.2 on_target_time

```
double PID::pid_config_t::on_target_time
```

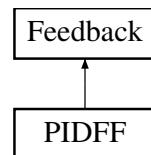
the time in seconds that we have to be on target for to say we are officially at the target

The documentation for this struct was generated from the following file:

- include/utils/controls/pid.h

5.65 PIDFF Class Reference

Inheritance diagram for PIDFF:



Public Member Functions

- **PIDFF** ([PID::pid_config_t](#) &pid_cfg, [FeedForward::ff_config_t](#) &ff_cfg)
- void [init](#) (double start_pt, double set_pt, double start_vel, double end_vel) override
- void [set_target](#) (double set_pt)
- double [get_target](#) () const
- double [get_sensor_val](#) () const
- double [update](#) (double val) override
- double [update](#) (double val, double vel_setpt, double a_setpt=0)
- double [get](#) () override
- void [set_limits](#) (double lower, double upper) override
- bool [is_on_target](#) () override
- void [reset](#) ()

Public Attributes

- [PID pid](#)

5.65.1 Member Function Documentation

5.65.1.1 get()

```
double PIDFF::get ( ) [override], [virtual]
```

Returns

the last saved result from the feedback controller

Implements [Feedback](#).

5.65.1.2 init()

```
void PIDFF::init (
    double start_pt,
    double set_pt,
    double start_vel,
    double end_vel ) [override], [virtual]
```

Initialize the feedback controller for a movement

Parameters

<i>start_pt</i>	the current sensor value
<i>set_pt</i>	where the sensor value should be
<i>start_vel</i>	the current rate of change of the sensor value
<i>end_vel</i>	the desired ending rate of change of the sensor value

Initialize the feedback controller for a movement

Parameters

<i>start←_pt</i>	the current sensor value
<i>set_pt</i>	where the sensor value should be

Implements [Feedback](#).

5.65.1.3 is_on_target()

```
bool PIDFF::is_on_target ( ) [override], [virtual]
```

Returns

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

5.65.1.4 set_limits()

```
void PIDFF::set_limits (
    double lower,
    double upper ) [override], [virtual]
```

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied.

Parameters

<i>lower</i>	Upper limit
<i>upper</i>	Lower limit

Implements [Feedback](#).

5.65.1.5 set_target()

```
void PIDFF::set_target (
    double set_pt )
```

Set the target of the [PID](#) loop

Parameters

<i>setpt</i>	Setpoint / target value
--------------	-------------------------

5.65.1.6 update() [1/2]

```
double PIDFF::update (
    double val ) [override], [virtual]
```

Iterate the feedback loop once with an updated sensor value. Only kS for feedforward will be applied.

Parameters

<i>val</i>	value from the sensor
------------	-----------------------

Returns

feedback loop result

Implements [Feedback](#).

5.65.1.7 update() [2/2]

```
double PIDFF::update (
    double val,
    double vel_setpt,
    double a_setpt = 0 )
```

Iterate the feedback loop once with an updated sensor value

Parameters

<i>val</i>	value from the sensor
<i>vel_setpt</i>	Velocity for feedforward
<i>a_setpt</i>	Acceleration for feedforward

Returns

feedback loop result

The documentation for this class was generated from the following files:

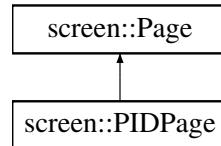
- include/utils/controls/pidff.h
- src/utils/controls/pidff.cpp

5.66 screen::PIDPage Class Reference

[PIDPage](#) provides a way to tune a pid controller on the screen.

```
#include <screen.h>
```

Inheritance diagram for screen::PIDPage:



Public Member Functions

- [PIDPage](#) (`PID &pid, std::string name, std::function< void(void)> onchange=[]() {}`)
Create a [PIDPage](#).
- [PIDPage](#) (`PIDFF &pidff, std::string name, std::function< void(void)> onchange=[]() {}`)
- void [update](#) (bool was_pressed, int x, int y) override
- void [draw](#) (vex::brain::lcd &, bool first_draw, unsigned int frame_number) override

5.66.1 Detailed Description

[PIDPage](#) provides a way to tune a pid controller on the screen.

5.66.2 Constructor & Destructor Documentation

5.66.2.1 PIDPage()

```
screen::PIDPage::PIDPage (
    PID & pid,
    std::string name,
    std::function< void(void)> onchange = []() {} )
```

Create a [PIDPage](#).

Parameters

<code>pid</code>	the pid controller we're changing
<code>name</code>	a name to recognize this pid controller if we've got multiple pid screens
<code>onchange</code>	a function that is called when a tuning parameter is changed. If you need to update stuff on that change register a handler here

5.66.3 Member Function Documentation

5.66.3.1 draw()

```
void screen::PIDPage::draw (
    vex::brain::lcd & scr,
    bool first_draw,
    unsigned int frame_number ) [override], [virtual]
```

See also

[Page::draw](#)

Reimplemented from [screen::Page](#).

5.66.3.2 update()

```
void screen::PIDPage::update (
    bool was_pressed,
    int x,
    int y ) [override], [virtual]
```

See also

[Page::update](#)

Reimplemented from [screen::Page](#).

The documentation for this class was generated from the following files:

- include/subsystems/screen.h
- src/subsystems/screen.cpp

5.67 plm_frame_t Struct Reference

Public Attributes

- double **time**
- unsigned int **width**
- unsigned int **height**
- [plm_plane_t](#) **y**
- [plm_plane_t](#) **cr**
- [plm_plane_t](#) **cb**

The documentation for this struct was generated from the following file:

- include/subsystems/fun/pl_mpeg.h

5.68 plm_packet_t Struct Reference

Public Attributes

- int **type**
- double **pts**
- size_t **length**
- uint8_t * **data**

The documentation for this struct was generated from the following file:

- include/subsystems/fun/pl_mpeg.h

5.69 plm_plane_t Struct Reference

Public Attributes

- unsigned int **width**
- unsigned int **height**
- uint8_t * **data**

The documentation for this struct was generated from the following file:

- include/subsystems/fun/pl_mpeg.h

5.70 plm_samples_t Struct Reference

Public Attributes

- double **time**
- unsigned int **count**
- float **interleaved** [PLM_AUDIO_SAMPLES_PER_FRAME *2]

The documentation for this struct was generated from the following file:

- include/subsystems/fun/pl_mpeg.h

5.71 point_t Struct Reference

```
#include <geometry.h>
```

Public Member Functions

- double `dist (const point_t other) const`
- `point_t operator+ (const point_t &other) const`
- `point_t operator- (const point_t &other) const`
- `point_t operator* (double s) const`
- `point_t operator/ (double s) const`
- `point_t operator- () const`
- `point_t operator+ () const`
- bool `operator== (const point_t &rhs)`

Public Attributes

- double `x`
the x position in space
- double `y`
the y position in space

5.71.1 Detailed Description

Data structure representing an X,Y coordinate

5.71.2 Member Function Documentation

5.71.2.1 dist()

```
double point_t::dist (
    const point_t other ) const [inline]
```

dist calculates the euclidian distance between this point and another point using the pythagorean theorem

Parameters

<code>other</code>	the point to measure the distance from
--------------------	--

Returns

the euclidian distance between this and other

5.71.2.2 operator+()

```
point_t point_t::operator+ (
    const point_t & other ) const [inline]
```

Vector2D addition operation on points

Parameters

<code>other</code>	the point to add on to this
--------------------	-----------------------------

Returns

`this + other (this.x + other.x, this.y + other.y)`

5.71.2.3 operator-()

```
point_t point_t::operator- (
    const point_t & other ) const [inline]
```

Vector2D subtraction operation on points

Parameters

<code>other</code>	the <code>point_t</code> to subtract from this
--------------------	--

Returns

`this - other (this.x - other.x, this.y - other.y)`

The documentation for this struct was generated from the following file:

- include/utils/geometry.h

5.72 pose_t Struct Reference

```
#include <geometry.h>
```

Public Member Functions

- `point_t get_point ()`

Public Attributes

- `double x`
x position in the world
- `double y`
y position in the world
- `double rot`
rotation in the world

5.72.1 Detailed Description

Describes a single position and rotation

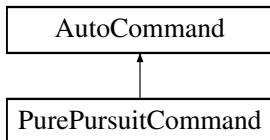
The documentation for this struct was generated from the following file:

- include/utils/geometry.h

5.73 PurePursuitCommand Class Reference

```
#include <drive_commands.h>
```

Inheritance diagram for PurePursuitCommand:



Public Member Functions

- [PurePursuitCommand \(TankDrive &drive_sys, Feedback &feedback, PurePursuit::Path path, directionType dir, double max_speed=1, double end_speed=0\)](#)
- [bool run \(\) override](#)
- [void on_timeout \(\) override](#)

Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand * withTimeout \(double t_seconds\)](#)
- [AutoCommand * withCancelCondition \(Condition *true_to_end\)](#)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- [double timeout_seconds = default_timeout](#)
- [Condition * true_to_end = nullptr](#)

Static Public Attributes inherited from [AutoCommand](#)

- [static constexpr double default_timeout = 10.0](#)

5.73.1 Detailed Description

Autocommand wrapper class for pure pursuit function in the [TankDrive](#) class

5.73.2 Constructor & Destructor Documentation

5.73.2.1 PurePursuitCommand()

```
PurePursuitCommand::PurePursuitCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    PurePursuit::Path path,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

Construct a Pure Pursuit [AutoCommand](#)

Parameters

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>feedback</i>	The feedback controller determining speed
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)

5.73.3 Member Function Documentation

5.73.3.1 on_timeout()

```
void PurePursuitCommand::on_timeout ( ) [override], [virtual]
```

Reset the drive system when it times out

Reimplemented from [AutoCommand](#).

5.73.3.2 run()

```
bool PurePursuitCommand::run ( ) [override], [virtual]
```

Direct call to [TankDrive::pure_pursuit](#)

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command_structure/drive_commands.h
- src/utils/command_structure/drive_commands.cpp

5.74 Rect Struct Reference

Public Member Functions

- [`point_t dimensions \(\) const`](#)
- [`point_t center \(\) const`](#)
- [`double width \(\) const`](#)
- [`double height \(\) const`](#)
- [`bool contains \(point_t p\) const`](#)

Static Public Member Functions

- static `Rect from_min_and_size (point_t min, point_t size)`

Public Attributes

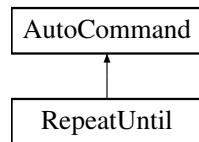
- `point_t min`
- `point_t max`

The documentation for this struct was generated from the following file:

- include/utils/geometry.h

5.75 RepeatUntil Class Reference

Inheritance diagram for RepeatUntil:



Public Member Functions

- `RepeatUntil (InOrder cmd, size_t repeats)`
RepeatUntil that runs a fixed number of times.
- `RepeatUntil (InOrder cmd, Condition *true_to_end)`
RepeatUntil the condition.
- `bool run () override`
- `void on_timeout () override`

Public Member Functions inherited from `AutoCommand`

- `AutoCommand * withTimeout (double t_seconds)`
- `AutoCommand * withCancelCondition (Condition *true_to_end)`

Additional Inherited Members

Public Attributes inherited from `AutoCommand`

- `double timeout_seconds = default_timeout`
- `Condition * true_to_end = nullptr`

Static Public Attributes inherited from `AutoCommand`

- static constexpr double `default_timeout = 10.0`

5.75.1 Constructor & Destructor Documentation

5.75.1.1 RepeatUntil() [1/2]

```
RepeatUntil::RepeatUntil (
    InOrder cmds,
    size_t repeats )
```

[RepeatUntil](#) that runs a fixed number of times.

Parameters

<i>cmds</i>	the cmds to repeat
<i>repeats</i>	the number of repeats to do

5.75.1.2 RepeatUntil() [2/2]

```
RepeatUntil::RepeatUntil (
    InOrder cmds,
    Condition * true_to_end )
```

[RepeatUntil](#) the condition.

Parameters

<i>cmds</i>	the cmds to run
<i>true_to_end</i>	we will repeat until <i>true_or_end</i> .test() returns true

5.75.2 Member Function Documentation**5.75.2.1 on_timeout()**

```
void RepeatUntil::on_timeout ( ) [override], [virtual]
```

What to do if we timeout instead of finishing. timeout is specified by the timeout seconds in the constructor

Reimplemented from [AutoCommand](#).

5.75.2.2 run()

```
bool RepeatUntil::run ( ) [override], [virtual]
```

Executes the command Overridden by child classes

Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command_structure/auto_command.h
- src/utils/command_structure/auto_command.cpp

5.76 robot_specs_t Struct Reference

```
#include <robot_specs.h>
```

Public Attributes

- double **robot_radius**
if you were to draw a circle with this radius, the robot would be entirely contained within it
- double **odom_wheel_diam**
the diameter of the wheels used for
- double **odom_gear_ratio**
the ratio of the odometry wheel to the encoder reading odometry data
- double **dist_between_wheels**
the distance between centers of the central drive wheels
- double **drive_correction_cutoff**
the distance at which to stop trying to turn towards the target. If we are less than this value, we can continue driving forward to minimize our distance but will not try to spin around to point directly at the target
- **Feedback * drive_feedback**
the default feedback for autonomous driving
- **Feedback * turn_feedback**
the defualt feedback for autonomous turning
- **PID::pid_config_t correction_pid**
the pid controller to keep the robot driving in as straight a line as possible

5.76.1 Detailed Description

Main robot characterization struct. This will be passed to all the major subsystems that require info about the robot. All distance measurements are in inches.

The documentation for this struct was generated from the following file:

- include/robot_specs.h

5.77 screen::ScreenData Struct Reference

The [ScreenData](#) class holds the data that will be passed to the screen thread you probably shouldnt have to use it.

Public Member Functions

- **ScreenData** (const std::vector< [Page](#) * > &m_pages, int m_page, vex::brain::lcd &m_screen)

Public Attributes

- std::vector< [Page](#) * > **pages**
- int **page** = 0
- vex::brain::lcd **screen**

5.77.1 Detailed Description

The `ScreenData` class holds the data that will be passed to the screen thread you probably shouldnt have to use it.

The documentation for this struct was generated from the following file:

- `src/subsystems/screen.cpp`

5.78 screen::ScreenRect Struct Reference

Public Attributes

- `uint32_t x1`
- `uint32_t y1`
- `uint32_t x2`
- `uint32_t y2`

The documentation for this struct was generated from the following file:

- `include/subsystems/screen.h`

5.79 Serializer Class Reference

Serializes Arbitrary data to a file on the SD Card.

```
#include <serializer.h>
```

Public Member Functions

- `~Serializer ()`
Save and close upon destruction (bc of vex, this doesnt always get called when the program ends. To be sure, call `save_to_disk`)
- `Serializer (const std::string &filename, bool flush_always=true)`
create a `Serializer`
- `void save_to_disk () const`
saves current `Serializer` state to disk
- `void set_int (const std::string &name, int i)`
Setters - not saved until `save_to_disk` is called.
- `void set_bool (const std::string &name, bool b)`
sets a bool by the name of name to b. If `flush_always == true`, this will save to the sd card
- `void set_double (const std::string &name, double d)`
sets a double by the name of name to d. If `flush_always == true`, this will save to the sd card
- `void set_string (const std::string &name, std::string str)`
sets a string by the name of name to s. If `flush_always == true`, this will save to the sd card
- `int int_or (const std::string &name, int otherwise)`
gets a value stored in the serializer. If not found, sets the value to otherwise
- `bool bool_or (const std::string &name, bool otherwise)`
gets a value stored in the serializer. If not, sets the value to otherwise
- `double double_or (const std::string &name, double otherwise)`
gets a value stored in the serializer. If not, sets the value to otherwise
- `std::string string_or (const std::string &name, std::string otherwise)`
gets a value stored in the serializer. If not, sets the value to otherwise

5.79.1 Detailed Description

Serializes Arbitrary data to a file on the SD Card.

5.79.2 Constructor & Destructor Documentation

5.79.2.1 Serializer()

```
Serializer::Serializer (
    const std::string & filename,
    bool flush_always = true ) [inline], [explicit]
```

create a [Serializer](#)

Parameters

<i>filename</i>	the file to read from. If filename does not exist we will create that file
<i>flush_always</i>	If true, after every write flush to a file. If false, you are responsible for calling <code>save_to_disk</code>

5.79.3 Member Function Documentation

5.79.3.1 bool_or()

```
bool Serializer::bool_or (
    const std::string & name,
    bool otherwise )
```

gets a value stored in the serializer. If not, sets the value to otherwise

Parameters

<i>name</i>	name of value
<i>otherwise</i>	value if the name is not specified

Returns

the value if found or otherwise

5.79.3.2 double_or()

```
double Serializer::double_or (
    const std::string & name,
    double otherwise )
```

gets a value stored in the serializer. If not, sets the value to otherwise

Parameters

<i>name</i>	name of value
<i>otherwise</i>	value if the name is not specified

Returns

the value if found or otherwise

5.79.3.3 int_or()

```
int Serializer::int_or (
    const std::string & name,
    int otherwise )
```

gets a value stored in the serializer. If not found, sets the value to otherwise

Getters Return value if it exists in the serializer

Parameters

<i>name</i>	name of value
<i>otherwise</i>	value if the name is not specified

Returns

the value if found or otherwise

5.79.3.4 save_to_disk()

```
void Serializer::save_to_disk ( ) const
```

saves current [Serializer](#) state to disk

forms data bytes then saves to filename this was openned with

5.79.3.5 set_bool()

```
void Serializer::set_bool (
    const std::string & name,
    bool b )
```

sets a bool by the name of name to b. If flush_always == true, this will save to the sd card

Parameters

<i>name</i>	name of bool
<i>b</i>	value of bool

5.79.3.6 set_double()

```
void Serializer::set_double (
    const std::string & name,
    double d )
```

sets a double by the name of name to d. If flush_always == true, this will save to the sd card

Parameters

<i>name</i>	name of double
<i>d</i>	value of double

5.79.3.7 set_int()

```
void Serializer::set_int (
    const std::string & name,
    int i )
```

Setters - not saved until save_to_disk is called.

sets an integer by the name of name to i. If flush_always == true, this will save to the sd card

Parameters

<i>name</i>	name of integer
<i>i</i>	value of integer

5.79.3.8 set_string()

```
void Serializer::set_string (
    const std::string & name,
    std::string str )
```

sets a string by the name of name to s. If flush_always == true, this will save to the sd card

Parameters

<i>name</i>	name of string
<i>i</i>	value of string

5.79.3.9 string_or()

```
std::string Serializer::string_or (
    const std::string & name,
    std::string otherwise )
```

gets a value stored in the serializer. If not, sets the value to otherwise

Parameters

<i>name</i>	name of value
<i>otherwise</i>	value if the name is not specified

Returns

the value if found or otherwise

The documentation for this class was generated from the following files:

- include/utils/serializer.h
- src/utils/serializer.cpp

5.80 screen::SizedWidget Struct Reference

Public Attributes

- int **size**
- [WidgetConfig](#) & **widget**

The documentation for this struct was generated from the following file:

- include/subsystems/screen.h

5.81 SliderCfg Struct Reference

Public Attributes

- double & **val**
- double **min**
- double **max**

The documentation for this struct was generated from the following file:

- include/subsystems/layout.h

5.82 screen::SliderConfig Struct Reference

Public Attributes

- double & **val**
- double **low**
- double **high**

The documentation for this struct was generated from the following file:

- include/subsystems/screen.h

5.83 screen::SliderWidget Class Reference

Widget that updates a double value. Updates by reference so watch out for race conditions cuz the screen stuff lives on another thread.

```
#include <screen.h>
```

Public Member Functions

- **SliderWidget** (double &val, double low, double high, **Rect** rect, std::string name)
Creates a slider widget.
- bool **update** (bool was_pressed, int x, int y)
responds to user input
- void **draw** (vex::brain::lcd &, bool first_draw, unsigned int frame_number)
Page::draws the slide to the screen

5.83.1 Detailed Description

Widget that updates a double value. Updates by reference so watch out for race conditions cuz the screen stuff lives on another thread.

5.83.2 Constructor & Destructor Documentation

5.83.2.1 SliderWidget()

```
screen::SliderWidget::SliderWidget (
    double & val,
    double low,
    double high,
    Rect rect,
    std::string name ) [inline]
```

Creates a slider widget.

Parameters

<i>val</i>	reference to the value to modify
<i>low</i>	minimum value to go to
<i>high</i>	maximum value to go to
<i>rect</i>	rect to draw it
<i>name</i>	name of the value

5.83.3 Member Function Documentation

5.83.3.1 update()

```
bool screen::SliderWidget::update (
```

```
    bool was_pressed,
    int x,
    int y )
```

responds to user input

Parameters

<code>was_pressed</code>	if the screen is pressed
<code>x</code>	x position if the screen was pressed
<code>y</code>	y position if the screen was pressed

Returns

true if the value updated

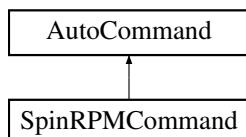
The documentation for this class was generated from the following files:

- include/subsystems/screen.h
- src/subsystems/screen.cpp

5.84 SpinRPMCommand Class Reference

```
#include <flywheel_commands.h>
```

Inheritance diagram for SpinRPMCommand:



Public Member Functions

- [SpinRPMCommand \(Flywheel &flywheel, int rpm\)](#)
- [bool run \(\) override](#)

Public Member Functions inherited from [AutoCommand](#)

- [virtual void on_timeout \(\)](#)
- [AutoCommand * withTimeout \(double t_seconds\)](#)
- [AutoCommand * withCancelCondition \(Condition *true_to_end\)](#)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- [double timeout_seconds = default_timeout](#)
- [Condition * true_to_end = nullptr](#)

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default_timeout** = 10.0

5.84.1 Detailed Description

File: [flywheel_commands.h](#) Desc: [insert meaningful desc] [AutoCommand](#) wrapper class for the spin_rpm function in the [Flywheel](#) class

5.84.2 Constructor & Destructor Documentation

5.84.2.1 SpinRPMCommand()

```
SpinRPMCommand::SpinRPMCommand (
    Flywheel & flywheel,
    int rpm )
```

Construct a SpinRPM Command

Parameters

<i>flywheel</i>	the flywheel sys to command
<i>rpm</i>	the rpm that we should spin at

File: [flywheel_commands.cpp](#) Desc: [insert meaningful desc]

5.84.3 Member Function Documentation

5.84.3.1 run()

```
bool SpinRPMCommand::run ( ) [override], [virtual]
```

Run spin_manual Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command_structure/flywheel_commands.h
- src/utils/command_structure/flywheel_commands.cpp

5.85 PurePursuit::spline Struct Reference

```
#include <pure_pursuit.h>
```

Public Member Functions

- double **getY** (double x)

Public Attributes

- double **a**
- double **b**
- double **c**
- double **d**
- double **x_start**
- double **x_end**

5.85.1 Detailed Description

Represents a piece of a cubic spline with $s(x) = a(x-x_i)^3 + b(x-x_i)^2 + c(x-x_i) + d$. The **x_start** and **x_end** shows where the equation is valid.

The documentation for this struct was generated from the following file:

- include/utils/pure_pursuit.h

5.86 StateMachine< System, IDType, Message, delay_ms, do_log >::State Struct Reference

```
#include <state_machine.h>
```

Public Member Functions

- virtual void **entry** (System &)
- virtual **MaybeMessage** **work** (System &)
- virtual void **exit** (System &)
- virtual **State** * **respond** (System &s, Message m)=0
- virtual IDType **id** () const =0

5.86.1 Detailed Description

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log = false>
struct StateMachine< System, IDType, Message, delay_ms, do_log >::State
```

Abstract class that all states for this machine must inherit from. States MUST override **respond()** and **id()** in order to function correctly (the compiler won't have it any other way)

The documentation for this struct was generated from the following file:

- include/utils/state_machine.h

5.87 StateMachine< System, IDType, Message, delay_ms, do_log > Class Template Reference

State Machine :)))))) A fun fun way of controlling stateful subsystems - used in the 2023-2024 Over Under game for our overly complex intake-cata subsystem (see there for an example) The statemachine runs in a background thread and a user thread can interact with it through current_state and send_message.

```
#include <state_machine.h>
```

Classes

- class [MaybeMessage](#)

MaybeMessage a message of Message type or nothing [MaybeMessage m = {};](#) // empty [MaybeMessage m = Message::EnumField1.](#)

- struct [State](#)

Public Types

- using [thread_data](#) = std::pair<[State](#) *, [StateMachine](#) *>

Public Member Functions

- [StateMachine \(State *initial\)](#)

Construct a state machine and immediately start running it.

- [IDType current_state \(\) const](#)

retrieve the current state of the state machine. This is safe to call from external threads

- void [send_message \(Message msg\)](#)

send a message to the state machine from outside

5.87.1 Detailed Description

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log = false>
class StateMachine< System, IDType, Message, delay_ms, do_log >
```

State Machine :)))))) A fun fun way of controlling stateful subsystems - used in the 2023-2024 Over Under game for our overly complex intake-cata subsystem (see there for an example) The statemachine runs in a background thread and a user thread can interact with it through current_state and send_message.

Designwise: the System class should hold onto any motors, feedback controllers, etc that are persistent in the system States themselves should hold any data that *only* that state needs. For example if a state should be exited after a certain amount of time, it should hold a timer rather than the System holding that timer. (see Junder from 2024 for an example of this design)

Template Parameters

System	The system that this is the base class of class Thing : public StateMachine<Thing> @tparam IDType The ID enum that recognizes states. Hint hint, use an enum class`
Message	the message enum that a state or an outside can send and that states respond to
delay_ms	the delay to wait between each state processing to allow other threads to work
do_log	if you want print statements describing incoming messages and current states. If true, it is expected that IDType and Message have a function called to_string that takes them as its only parameter and returns a std::string

5.87.2 Constructor & Destructor Documentation

5.87.2.1 StateMachine()

```
template<typename System , typename IDType , typename Message , int32_t delay_ms, bool do_log
= false>
StateMachine< System, IDType, Message, delay_ms, do_log >::StateMachine (
    State * initial ) [inline]
```

Construct a state machine and immediately start running it.

Parameters

<i>initial</i>	the state that the machine will begin in
----------------	--

5.87.3 Member Function Documentation

5.87.3.1 current_state()

```
template<typename System , typename IDType , typename Message , int32_t delay_ms, bool do_log
= false>
IDType StateMachine< System, IDType, Message, delay_ms, do_log >::current_state ( ) const
[inline]
```

retrieve the current state of the state machine. This is safe to call from external threads

Returns

the current state

5.87.3.2 send_message()

```
template<typename System , typename IDType , typename Message , int32_t delay_ms, bool do_log
= false>
void StateMachine< System, IDType, Message, delay_ms, do_log >::send_message (
    Message msg ) [inline]
```

send a message to the state machine from outside

Parameters

<i>msg</i>	the message to send This is safe to call from external threads
------------	--

The documentation for this class was generated from the following file:

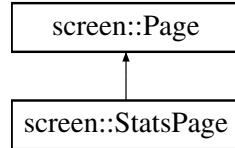
- include/utils/state_machine.h

5.88 screen::StatsPage Class Reference

Draws motor stats and battery stats to the screen.

```
#include <screen.h>
```

Inheritance diagram for screen::StatsPage:



Public Member Functions

- **StatsPage** (std::map< std::string, vex::motor & > motors)
Creates a stats page.
- void **update** (bool was_pressed, int x, int y) override
- void **draw** (vex::brain::lcd &, bool first_draw, unsigned int frame_number) override

5.88.1 Detailed Description

Draws motor stats and battery stats to the screen.

5.88.2 Constructor & Destructor Documentation

5.88.2.1 StatsPage()

```
screen::StatsPage::StatsPage (
    std::map< std::string, vex::motor & > motors )
```

Creates a stats page.

Parameters

<i>motors</i>	a map of string to motor that we want to draw on this page
---------------	--

5.88.3 Member Function Documentation

5.88.3.1 draw()

```
void screen::StatsPage::draw (
    vex::brain::lcd & scr,
    bool first_draw,
    unsigned int frame_number ) [override], [virtual]
```

See also

[Page::draw](#)

Reimplemented from [screen::Page](#).

5.88.3.2 update()

```
void screen::StatsPage::update (
    bool was_pressed,
    int x,
    int y ) [override], [virtual]
```

See also

[Page::update](#)

Reimplemented from [screen::Page](#).

The documentation for this class was generated from the following files:

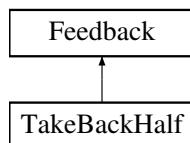
- include/subsystems/screen.h
- src/subsystems/screen.cpp

5.89 TakeBackHalf Class Reference

A velocity controller.

```
#include <take_back_half.h>
```

Inheritance diagram for TakeBackHalf:



Public Member Functions

- **TakeBackHalf** (double [TBH_gain](#), double [first_cross_split](#), double [on_target_threshold](#))
- void [init](#) (double [start_pt](#), double [set_pt](#), double, double)
- double [update](#) (double [val](#)) override
- double [get](#) () override
- void [set_limits](#) (double [lower](#), double [upper](#)) override
- bool [is_on_target](#) () override

Public Attributes

- double **TBH_gain**
tuned parameter
- double **first_cross_split**

5.89.1 Detailed Description

A velocity controller.

Warning

If you try to use this as a position controller, it will fail.

5.89.2 Member Function Documentation

5.89.2.1 get()

```
double TakeBackHalf::get ( ) [override], [virtual]
```

Returns

the last saved result from the feedback controller

Implements [Feedback](#).

5.89.2.2 init()

```
void TakeBackHalf::init (
    double start_pt,
    double set_pt,
    double ,
    double ) [virtual]
```

Initialize the feedback controller for a movement

Parameters

<i>start_pt</i>	the current sensor value
<i>set_pt</i>	where the sensor value should be
<i>start_vel</i>	Movement starting velocity (IGNORED)
<i>end_vel</i>	Movement ending velocity (IGNORED)

Implements [Feedback](#).

5.89.2.3 is_on_target()

```
bool TakeBackHalf::is_on_target ( ) [override], [virtual]
```

Returns

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

5.89.2.4 set_limits()

```
void TakeBackHalf::set_limits (
    double lower,
    double upper ) [override], [virtual]
```

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied.

Parameters

<i>lower</i>	Upper limit
<i>upper</i>	Lower limit

Implements [Feedback](#).

5.89.2.5 update()

```
double TakeBackHalf::update (
    double val ) [override], [virtual]
```

Iterate the feedback loop once with an updated sensor value

Parameters

<i>val</i>	value from the sensor
------------	-----------------------

Returns

feedback loop result

Implements [Feedback](#).

The documentation for this class was generated from the following files:

- include/utils/controls/take_back_half.h
- src/utils/controls/take_back_half.cpp

5.90 TankDrive Class Reference

```
#include <tank_drive.h>
```

Public Types

- enum class `BrakeType` { `None` , `ZeroVelocity` , `Smart` }

Public Member Functions

- `TankDrive` (`motor_group &left_motors, motor_group &right_motors, robot_specs_t &config, OdometryBase *odom=NULL`)
- `AutoCommand * DriveToPointCmd` (`point_t pt, vex::directionType dir=vex::forward, double max_speed=1.0, double end_speed=0.0`)
- `AutoCommand * DriveToPointCmd` (`Feedback &fb, point_t pt, vex::directionType dir=vex::forward, double max_speed=1.0, double end_speed=0.0`)
- `AutoCommand * DriveForwardCmd` (`double dist, vex::directionType dir=vex::forward, double max_speed=1.0, double end_speed=0.0`)
- `AutoCommand * DriveForwardCmd` (`Feedback &fb, double dist, vex::directionType dir=vex::forward, double max_speed=1.0, double end_speed=0.0`)
- `AutoCommand * TurnToHeadingCmd` (`double heading, double max_speed=1.0, double end_speed=0.0`)
- `AutoCommand * TurnToHeadingCmd` (`Feedback &fb, double heading, double max_speed=1.0, double end_speed=0.0`)
- `AutoCommand * TurnToPointCmd` (`double x, double y, vex::directionType dir=vex::directionType::fwd, double max_speed=1.0, double end_speed=0.0`)
- `AutoCommand * TurnDegreesCmd` (`double degrees, double max_speed=1.0, double start_speed=0.0`)
- `AutoCommand * TurnDegreesCmd` (`Feedback &fb, double degrees, double max_speed=1.0, double end_speed=0.0`)
- `AutoCommand * PurePursuitCmd` (`PurePursuit::Path path, directionType dir, double max_speed=1, double end_speed=0`)
- `AutoCommand * PurePursuitCmd` (`Feedback &feedback, PurePursuit::Path path, directionType dir, double max_speed=1, double end_speed=0`)
- `Condition * DriveStalledCondition` (`double stall_time`)
- `AutoCommand * DriveTankCmd` (`double left, double right`)
- `void stop ()`
- `void drive_tank` (`double left, double right, int power=1, BrakeType bt=BrakeType::None`)
- `void drive_tank_raw` (`double left, double right`)
- `void drive_arcade` (`double forward_back, double left_right, int power=1, BrakeType bt=BrakeType::None`)
- `bool drive_forward` (`double inches, directionType dir, Feedback &feedback, double max_speed=1, double end_speed=0`)
- `bool drive_forward` (`double inches, directionType dir, double max_speed=1, double end_speed=0`)
- `bool turn_degrees` (`double degrees, Feedback &feedback, double max_speed=1, double end_speed=0`)
- `bool turn_degrees` (`double degrees, double max_speed=1, double end_speed=0`)
- `bool drive_to_point` (`double x, double y, vex::directionType dir, Feedback &feedback, double max_speed=1, double end_speed=0`)
- `bool drive_to_point` (`double x, double y, vex::directionType dir, double max_speed=1, double end_speed=0`)
- `bool turn_to_heading` (`double heading_deg, Feedback &feedback, double max_speed=1, double end_speed=0`)
- `bool turn_to_heading` (`double heading_deg, double max_speed=1, double end_speed=0`)
- `void reset_auto ()`
- `bool pure_pursuit` (`PurePursuit::Path path, directionType dir, Feedback &feedback, double max_speed=1, double end_speed=0`)
- `bool pure_pursuit` (`PurePursuit::Path path, directionType dir, double max_speed=1, double end_speed=0`)

Static Public Member Functions

- static double `modify_inputs` (`double input, int power=2`)

5.90.1 Detailed Description

[TankDrive](#) is a class to run a tank drive system. A tank drive system, sometimes called differential drive, has a motor (or group of synchronized motors) on the left and right side

5.90.2 Member Enumeration Documentation

5.90.2.1 BrakeType

```
enum class TankDrive::BrakeType [strong]
```

Enumerator

None	just send 0 volts to the motors
ZeroVelocity	try to bring the robot to rest. But don't try to hold position
Smart	bring the robot to rest and once it's stopped, try to hold that position

5.90.3 Constructor & Destructor Documentation

5.90.3.1 TankDrive()

```
TankDrive::TankDrive (
    motor_group & left_motors,
    motor_group & right_motors,
    robot_specs_t & config,
    OdometryBase * odom = NULL )
```

Create the [TankDrive](#) object

Parameters

<i>left_motors</i>	left side drive motors
<i>right_motors</i>	right side drive motors
<i>config</i>	the configuration specification defining physical dimensions about the robot. See robot_specs_t for more info
<i>odom</i>	an odometry system to track position and rotation. this is necessary to execute autonomous paths

5.90.4 Member Function Documentation

5.90.4.1 drive_arCADE()

```
void TankDrive::drive_arCADE (
    double forward_back,
    double left_right,
```

```
int power = 1,
BrakeType bt = BrakeType::None )
```

Drive the robot using arcade style controls. forward_back controls the linear motion, left_right controls the turning.

forward_back and left_right are in "percent": -1.0 -> 1.0

Parameters

<i>forward_back</i>	the percent to move forward or backward
<i>left_right</i>	the percent to turn left or right
<i>power</i>	modifies the input velocities left^power, right^power
<i>bt</i>	breaktype. What to do if the driver lets go of the sticks

Drive the robot using arcade style controls. forward_back controls the linear motion, left_right controls the turning.

left_motors and right_motors are in "percent": -1.0 -> 1.0

5.90.4.2 drive_forward() [1/2]

```
bool TankDrive::drive_forward (
    double inches,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

Autonomously drive the robot forward a certain distance

Parameters

<i>inches</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Autonomously drive the robot forward a certain distance

Parameters

<i>inches</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have finished driving to our point

5.90.4.3 drive_forward() [2/2]

```
bool TankDrive::drive_forward (
    double inches,
    directionType dir,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0 )
```

Use odometry to drive forward a certain distance using a custom feedback controller

Returns whether or not the robot has reached it's destination.

Parameters

<i>inches</i>	the distance to drive forward
<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the custom feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Returns

true when we have reached our target distance

Use odometry to drive forward a certain distance using a custom feedback controller

Returns whether or not the robot has reached it's destination.

Parameters

<i>inches</i>	the distance to drive forward
<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the custom feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

5.90.4.4 drive_tank()

```
void TankDrive::drive_tank (
    double left,
    double right,
    int power = 1,
    BrakeType bt = BrakeType::None )
```

Drive the robot using differential style controls. *left_motors* controls the left motors, *right_motors* controls the right motors.

left_motors and *right_motors* are in "percent": -1.0 -> 1.0

Parameters

<i>left</i>	the percent to run the left motors
<i>right</i>	the percent to run the right motors
<i>power</i>	modifies the input velocities left^power, right^power
<i>bt</i>	breaktype. What to do if the driver lets go of the sticks

5.90.4.5 drive_tank_raw()

```
void TankDrive::drive_tank_raw (
    double left,
    double right )
```

Drive the robot raw-ly

Parameters

<i>left</i>	the percent to run the left motors (-1, 1)
<i>right</i>	the percent to run the right motors (-1, 1)

5.90.4.6 drive_to_point() [1/2]

```
bool TankDrive::drive_to_point (
    double x,
    double y,
    vex::directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot. Here we use the default feedback controller from the drive_sys

Returns whether or not the robot has reached it's destination.

Parameters

<i>x</i>	the x position of the target
<i>y</i>	the y position of the target
<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot. Here we use the default feedback controller from the drive_sys

Returns whether or not the robot has reached it's destination.

Parameters

<i>x</i>	the x position of the target
<i>y</i>	the y position of the target
<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have reached our target point

5.90.4.7 drive_to_point() [2/2]

```
bool TankDrive::drive_to_point (
    double x,
    double y,
    vex::directionType dir,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0 )
```

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot.

Returns whether or not the robot has reached it's destination.

Parameters

<i>x</i>	the x position of the target
<i>y</i>	the y position of the target
<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot.

Returns whether or not the robot has reached it's destination.

Parameters

<i>x</i>	the x position of the target
<i>y</i>	the y position of the target
<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have reached our target point

5.90.4.8 modify_inputs()

```
double TankDrive::modify_inputs (
    double input,
    int power = 2 ) [static]
```

Create a curve for the inputs, so that drivers have more control at lower speeds. Curves are exponential, with the default being squaring the inputs.

Parameters

<i>input</i>	the input before modification
<i>power</i>	the power to raise input to

Returns

$\text{input}^{\wedge} \text{power}$ (accounts for negative inputs and odd numbered powers)

Modify the inputs from the controller by squaring / cubing, etc Allows for better control of the robot at slower speeds

Parameters

<i>input</i>	the input signal -1 -> 1
<i>power</i>	the power to raise the signal to

Returns

$\text{input}^{\wedge} \text{power}$ accounting for any sign issues that would arise with this naive solution

5.90.4.9 pure_pursuit() [1/2]

```
bool TankDrive::pure_pursuit (
    PurePursuit::Path path,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

Use the default drive feedback

Parameters

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Returns

True when the path is complete

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

Use the default drive feedback

Parameters

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)

Returns

True when the path is complete

5.90.4.10 pure_pursuit() [2/2]

```
bool TankDrive::pure_pursuit (
    PurePursuit::Path path,
    directionType dir,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0 )
```

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

Parameters

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>feedback</i>	The feedback controller determining speed
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Returns

True when the path is complete

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

Parameters

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>feedback</i>	The feedback controller determining speed
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)

Returns

True when the path is complete

5.90.4.11 reset_auto()

```
void TankDrive::reset_auto ( )
```

Reset the initialization for autonomous drive functions

5.90.4.12 stop()

```
void TankDrive::stop ( )
```

Stops rotation of all the motors using their "brake mode"

5.90.4.13 turn_degrees() [1/2]

```
bool TankDrive::turn_degrees (
    double degrees,
    double max_speed = 1,
    double end_speed = 0 )
```

Autonomously turn the robot X degrees to counterclockwise (negative for clockwise), with a maximum motor speed of percent_speed (-1.0 -> 1.0)

Uses the default turning feedback of the drive system.

Parameters

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Autonomously turn the robot X degrees to counterclockwise (negative for clockwise), with a maximum motor speed of percent_speed (-1.0 -> 1.0)

Uses the default turning feedback of the drive system.

Parameters

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Returns

true if we turned to target number of degrees

5.90.4.14 turn_degrees() [2/2]

```
bool TankDrive::turn_degrees (
    double degrees,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0 )
```

Autonomously turn the robot X degrees counterclockwise (negative for clockwise), with a maximum motor speed of percent_speed (-1.0 -> 1.0)

Uses PID + Feedforward for it's control.

Parameters

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power

Autonomously turn the robot X degrees to counterclockwise (negative for clockwise), with a maximum motor speed of percent_speed (-1.0 -> 1.0)

Uses the specified feedback for it's control.

Parameters

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have turned our target number of degrees

5.90.4.15 turn_to_heading() [1/2]

```
bool TankDrive::turn_to_heading (
    double heading_deg,
    double max_speed = 1,
    double end_speed = 0 )
```

Turn the robot in place to an exact heading relative to the field. 0 is forward. Uses the defualt turn feedback of the drive system

Parameters

<i>heading_deg</i>	the heading to which we will turn
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Turn the robot in place to an exact heading relative to the field. 0 is forward. Uses the default turn feedback of the drive system

Parameters

<i>heading_deg</i>	the heading to which we will turn
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have reached our target heading

5.90.4.16 turn_to_heading() [2/2]

```
bool TankDrive::turn_to_heading (
    double heading_deg,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0 )
```

Turn the robot in place to an exact heading relative to the field. 0 is forward.

Parameters

<i>heading_deg</i>	the heading to which we will turn
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Turn the robot in place to an exact heading relative to the field. 0 is forward.

Parameters

<i>heading_deg</i>	the heading to which we will turn
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have reached our target heading

The documentation for this class was generated from the following files:

- include/subsystems/tank_drive.h
- src/subsystems/tank_drive.cpp

5.91 screen::TextConfig Struct Reference

Public Attributes

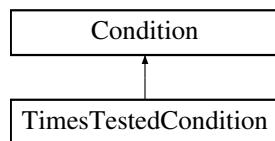
- std::function< std::string()> **text**

The documentation for this struct was generated from the following file:

- include/subsystems/screen.h

5.92 TimesTestedCondition Class Reference

Inheritance diagram for TimesTestedCondition:



Public Member Functions

- **TimesTestedCondition** (size_t N)
- bool **test** () override

Public Member Functions inherited from [Condition](#)

- [Condition * Or \(Condition *b\)](#)
- [Condition * And \(Condition *b\)](#)

5.92.1 Member Function Documentation

5.92.1.1 test()

```
bool TimesTestedCondition::test ( ) [inline], [override], [virtual]
```

Implements [Condition](#).

The documentation for this class was generated from the following file:

- include/utils/command_structure/auto_command.h

5.93 trapezoid_profile_segment_t Struct Reference

```
#include <trapezoid_profile.h>
```

Public Attributes

- double **pos_after**
1d position after this segment concludes
- double **vel_after**
1d velocity after this segment concludes
- double **accel**
1d acceleration during the segment
- double **duration**
duration of the segment

5.93.1 Detailed Description

`trapezoid_profile_segment_t` is a description of one constant acceleration segment of a trapezoid motion profile

The documentation for this struct was generated from the following file:

- include/utils/controls/trapezoid_profile.h

5.94 TrapezoidProfile Class Reference

```
#include <trapezoid_profile.h>
```

Public Member Functions

- **TrapezoidProfile** (double max_v, double accel)
Construct a new Trapezoid Profile object.
- **motion_t calculate** (double time_s, double pos_s)
Run the trapezoidal profile based on the time and distance that's elapsed.
- **motion_t calculate_time_based** (double time_s)
Run the trapezoidal profile based on the time that's elapsed.
- void **set_endpts** (double start, double end)
set_endpts defines a start and end position
- void **set_vel_endpts** (double start, double end)
set start and end velocities
- void **set_accel** (double accel)
set_accel sets the acceleration this profile will use (the left and right legs of the trapezoid)
- void **set_max_v** (double max_v)
sets the maximum velocity for the profile (the height of the top of the trapezoid)
- double **get_movement_time** () const
uses the kinematic equations to and specified accel and max_v to figure out how long moving along the profile would take
- double **get_max_v** () const
- double **get_accel** () const

5.94.1 Detailed Description

Trapezoid Profile

This is a motion profile defined by:

- maximum acceleration
- maximum velocity
- start position and velocity
- end position and velocity

Using this information, a parametric function is generated, with a period of acceleration, constant velocity, and deceleration. The velocity graph usually looks like a trapezoid, giving it its name.

If the maximum velocity is set high enough, this will become a S-curve profile, with only acceleration and deceleration.

If the initial velocity is in the wrong direction, the profile will first come to a stop, then continue a normal trapezoid profile.

If the initial velocity is higher than the maximum velocity, the profile will first try to achieve the maximum velocity.

If the end velocity is not achievable, the profile will try to get as close as possible. The end velocity must be in the direction of the end point.

This class is designed for use in properly modelling the motion of the robots to create a feedforward and target for **PID**. Acceleration and Maximum velocity should be measured on the robot and tuned down slightly to account for battery drop.

Here are the equations graphed for ease of understanding: <https://www.desmos.com/calculator/rkm3ivulyk>

Author

Ryan McGee

Date

7/12/2022

5.94.2 Constructor & Destructor Documentation

5.94.2.1 TrapezoidProfile()

```
TrapezoidProfile::TrapezoidProfile (
    double max_v,
    double accel )
```

Construct a new Trapezoid Profile object.

Parameters

<i>max_v</i>	Maximum velocity the robot can run at
<i>accel</i>	Maximum acceleration of the robot

5.94.3 Member Function Documentation**5.94.3.1 calculate()**

```
motion\_t TrapezoidProfile::calculate (
    double time_s,
    double pos_s )
```

Run the trapezoidal profile based on the time and distance that's elapsed.

Parameters

<i>time_s</i>	Time since start of movement
<i>pos_s</i>	The current position

Returns

[*motion_t*](#) Position, velocity and acceleration

5.94.3.2 calculate_time_based()

```
motion\_t TrapezoidProfile::calculate_time_based (
    double time_s )
```

Run the trapezoidal profile based on the time that's elapsed.

Parameters

<i>time_s</i>	Time since start of movement
---------------	------------------------------

Returns

[*motion_t*](#) Position, velocity and acceleration

5.94.3.3 get_movement_time()

```
double TrapezoidProfile::get_movement_time ( ) const
```

uses the kinematic equations to and specified accel and max_v to figure out how long moving along the profile would take

Returns

the time the path will take to travel

5.94.3.4 set_accel()

```
void TrapezoidProfile::set_accel (
    double accel )
```

`set_accel` sets the acceleration this profile will use (the left and right legs of the trapezoid)

Parameters

<code>accel</code>	the acceleration amount to use
--------------------	--------------------------------

5.94.3.5 set_endpts()

```
void TrapezoidProfile::set_endpts (
    double start,
    double end )
```

`set_endpts` defines a start and end position

Parameters

<code>start</code>	the starting position of the path
<code>end</code>	the ending position of the path

5.94.3.6 set_max_v()

```
void TrapezoidProfile::set_max_v (
    double max_v )
```

`sets the maximum velocity for the profile (the height of the top of the trapezoid)`

Parameters

<code>max_v</code>	the maximum velocity the robot can travel at
--------------------	--

5.94.3.7 set_vel_endpts()

```
void TrapezoidProfile::set_vel_endpts (
    double start,
    double end )
```

`set start and end velocities`

Parameters

<code>start</code>	the starting velocity of the path
<code>end</code>	the ending velocity of the path

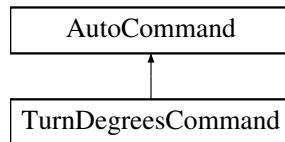
The documentation for this class was generated from the following files:

- include/utils/controls/trapezoid_profile.h
- src/utils/trapezoid_profile.cpp

5.95 TurnDegreesCommand Class Reference

```
#include <drive_commands.h>
```

Inheritance diagram for TurnDegreesCommand:



Public Member Functions

- `TurnDegreesCommand (TankDrive &drive_sys, Feedback &feedback, double degrees, double max_speed=1, double end_speed=0)`
- `bool run () override`
- `void on_timeout () override`

Public Member Functions inherited from [AutoCommand](#)

- `AutoCommand * withTimeout (double t_seconds)`
- `AutoCommand * withCancelCondition (Condition *true_to_end)`

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- `double timeout_seconds = default_timeout`
- `Condition * true_to_end = nullptr`

Static Public Attributes inherited from [AutoCommand](#)

- `static constexpr double default_timeout = 10.0`

5.95.1 Detailed Description

[AutoCommand](#) wrapper class for the turn_degrees function in the [TankDrive](#) class

5.95.2 Constructor & Destructor Documentation

5.95.2.1 TurnDegreesCommand()

```
TurnDegreesCommand::TurnDegreesCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    double degrees,
    double max_speed = 1,
    double end_speed = 0 )
```

Construct a [TurnDegreesCommand](#) Command

Parameters

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the turn
<i>degrees</i>	how many degrees to rotate
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

5.95.3 Member Function Documentation

5.95.3.1 on_timeout()

```
void TurnDegreesCommand::on_timeout ( ) [override], [virtual]
```

Cleans up drive system if we time out before finishing

reset the drive system if we timeout

Reimplemented from [AutoCommand](#).

5.95.3.2 run()

```
bool TurnDegreesCommand::run ( ) [override], [virtual]
```

Run turn_degrees Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

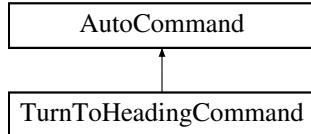
The documentation for this class was generated from the following files:

- include/utils/command_structure/drive_commands.h
- src/utils/command_structure/drive_commands.cpp

5.96 TurnToHeadingCommand Class Reference

```
#include <drive_commands.h>
```

Inheritance diagram for TurnToHeadingCommand:



Public Member Functions

- [TurnToHeadingCommand \(TankDrive &drive_sys, Feedback &feedback, double heading_deg, double speed=1, double end_speed=0\)](#)
- [bool run \(\) override](#)
- [void on_timeout \(\) override](#)

Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand * withTimeout \(double t_seconds\)](#)
- [AutoCommand * withCancelCondition \(Condition *true_to_end\)](#)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- double [timeout_seconds](#) = default_timeout
- [Condition * true_to_end](#) = nullptr

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default_timeout](#) = 10.0

5.96.1 Detailed Description

[AutoCommand](#) wrapper class for the turn_to_heading() function in the [TankDrive](#) class

5.96.2 Constructor & Destructor Documentation

5.96.2.1 TurnToHeadingCommand()

```
TurnToHeadingCommand::TurnToHeadingCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    double heading_deg,
    double max_speed = 1,
    double end_speed = 0 )
```

Construct a [TurnToHeadingCommand](#) Command

Parameters

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the drive
<i>heading_deg</i>	the heading to turn to in degrees
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

5.96.3 Member Function Documentation**5.96.3.1 on_timeout()**

```
void TurnToHeadingCommand::on_timeout ( ) [override], [virtual]
```

Cleans up drive system if we time out before finishing

reset the drive system if we don't hit our target

Reimplemented from [AutoCommand](#).

5.96.3.2 run()

```
bool TurnToHeadingCommand::run ( ) [override], [virtual]
```

Run turn_to_heading Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command_structure/drive_commands.h
- src/utils/command_structure/drive_commands.cpp

5.97 Vector2D Class Reference

```
#include <vector2d.h>
```

Public Member Functions

- [Vector2D \(double dir, double mag\)](#)
- [Vector2D \(point_t p\)](#)
- [double get_dir \(\) const](#)
- [double get_mag \(\) const](#)
- [double get_x \(\) const](#)
- [double get_y \(\) const](#)
- [Vector2D normalize \(\)](#)
- [point_t point \(\)](#)
- [Vector2D operator* \(const double &x\)](#)
- [Vector2D operator+ \(const Vector2D &other\)](#)
- [Vector2D operator- \(const Vector2D &other\)](#)

5.97.1 Detailed Description

`Vector2D` is an x,y pair Used to represent 2D locations on the field. It can also be treated as a direction and magnitude

5.97.2 Constructor & Destructor Documentation

5.97.2.1 Vector2D() [1/2]

```
Vector2D::Vector2D (
    double dir,
    double mag )
```

Construct a vector object.

Parameters

<code>dir</code>	Direction, in radians. 'forward' is 0, clockwise positive when viewed from the top.
<code>mag</code>	Magnitude.

5.97.2.2 Vector2D() [2/2]

```
Vector2D::Vector2D (
    point_t p )
```

Construct a vector object from a cartesian point.

Parameters

<code>p</code>	<code>point_t.x</code> , <code>point_t.y</code>
----------------	---

5.97.3 Member Function Documentation

5.97.3.1 get_dir()

```
double Vector2D::get_dir ( ) const
```

Get the direction of the vector, in radians. '0' is forward, clockwise positive when viewed from the top.

Use r2d() to convert.

Returns

the direction of the vector in radians

Get the direction of the vector, in radians. '0' is forward, clockwise positive when viewed from the top.

Use r2d() to convert.

5.97.3.2 get_mag()

```
double Vector2D::get_mag ( ) const
```

Returns

the magnitude of the vector

Get the magnitude of the vector

5.97.3.3 get_x()

```
double Vector2D::get_x ( ) const
```

Returns

the X component of the vector; positive to the right.

Get the X component of the vector; positive to the right.

5.97.3.4 get_y()

```
double Vector2D::get_y ( ) const
```

Returns

the Y component of the vector, positive forward.

Get the Y component of the vector, positive forward.

5.97.3.5 normalize()

```
Vector2D Vector2D::normalize ( )
```

Changes the magnitude of the vector to 1

Returns

the normalized vector

Changes the magnitude of the vector to 1

5.97.3.6 operator*()

```
Vector2D Vector2D::operator* (
    const double & x )
```

Scales a [Vector2D](#) by a scalar with the * operator

Parameters

x	the value to scale the vector by
---	----------------------------------

Returns

the this [Vector2D](#) scaled by x

5.97.3.7 operator+()

```
Vector2D Vector2D::operator+ (
    const Vector2D & other )
```

Add the components of two vectors together $\text{Vector2D} + \text{Vector2D} = (\text{this.x} + \text{other.x}, \text{this.y} + \text{other.y})$

Parameters

other	the vector to add to this
-------	---------------------------

Returns

the sum of the vectors

5.97.3.8 operator-()

```
Vector2D Vector2D::operator- (
    const Vector2D & other )
```

Subtract the components of two vectors together $\text{Vector2D} - \text{Vector2D} = (\text{this.x} - \text{other.x}, \text{this.y} - \text{other.y})$

Parameters

<i>other</i>	the vector to subtract from this
--------------	----------------------------------

Returns

the difference of the vectors

5.97.3.9 point()

`point_t Vector2D::point ()`

Returns a point from the vector

Returns

the point represented by the vector

Convert a direction and magnitude representation to an x, y representation

Returns

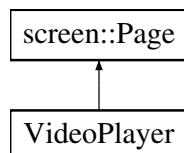
the x, y representation of the vector

The documentation for this class was generated from the following files:

- `include/utils/vector2d.h`
- `src/utils/vector2d.cpp`

5.98 VideoPlayer Class Reference

Inheritance diagram for VideoPlayer:

**Public Member Functions**

- void `update` (bool *was_pressed*, int *x*, int *y*) override
collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this Page (only drawn page gets touch updates))
- void `draw` (vex::brain::lcd &*screen*, bool *first_draw*, unsigned int *frame_number*) override
draw stored data to the screen (runs at 10 hz and only runs if this page is in front)

5.98.1 Member Function Documentation**5.98.1.1 draw()**

```

void VideoPlayer::draw (
    vex::brain::lcd & screen,
    bool first_draw,
    unsigned int frame_number ) [override], [virtual]
  
```

draw stored data to the screen (runs at 10 hz and only runs if this page is in front)

Parameters

<i>first_draw</i>	true if we just switched to this page
<i>frame_number</i>	frame of drawing we are on (basically an animation tick)

Reimplemented from [screen::Page](#).

5.98.1.2 update()

```
void VideoPlayer::update (
    bool was_pressed,
    int x,
    int y) [override], [virtual]
```

collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this Page (only drawn page gets touch updates))

Parameters

<i>was_pressed</i>	true if the screen has been pressed
<i>x</i>	x position of screen press (if the screen was pressed)
<i>y</i>	y position of screen press (if the screen was pressed)

Reimplemented from [screen::Page](#).

The documentation for this class was generated from the following files:

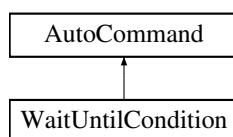
- include/subsystems/fun/video.h
- src/subsystems/fun/video.cpp

5.99 WaitUntilCondition Class Reference

Waits until the condition is true.

```
#include <auto_command.h>
```

Inheritance diagram for WaitUntilCondition:

**Public Member Functions**

- **WaitUntilCondition** ([Condition](#) *cond)
- bool **run** () override

Public Member Functions inherited from [AutoCommand](#)

- virtual void [on_timeout \(\)](#)
- [AutoCommand * withTimeout](#) (double t_seconds)
- [AutoCommand * withCancelCondition](#) ([Condition](#) *true_to_end)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- double [timeout_seconds](#) = default_timeout
- [Condition * true_to_end](#) = nullptr

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default_timeout](#) = 10.0

5.99.1 Detailed Description

Waits until the condition is true.

5.99.2 Member Function Documentation

5.99.2.1 run()

```
bool WaitUntilCondition::run ( ) [inline], [override], [virtual]
```

Executes the command Overridden by child classes

Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

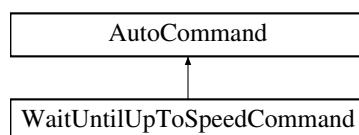
The documentation for this class was generated from the following file:

- include/utils/command_structure/auto_command.h

5.100 WaitUntilUpToSpeedCommand Class Reference

```
#include <flywheel_commands.h>
```

Inheritance diagram for WaitUntilUpToSpeedCommand:



Public Member Functions

- [WaitUntilUpToSpeedCommand \(`Flywheel` &`flywheel`, int `threshold_rpm`\)](#)
- bool `run ()` override

Public Member Functions inherited from [AutoCommand](#)

- virtual void `on_timeout ()`
- [AutoCommand * withTimeout \(double `t_seconds`\)](#)
- [AutoCommand * withCancelCondition \(`Condition` *`true_to_end`\)](#)

Additional Inherited Members

Public Attributes inherited from [AutoCommand](#)

- double `timeout_seconds` = `default_timeout`
- `Condition` * `true_to_end` = `nullptr`

Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double `default_timeout` = 10.0

5.100.1 Detailed Description

[AutoCommand](#) that listens to the [Flywheel](#) and waits until it is at its target speed +/- the specified threshold

5.100.2 Constructor & Destructor Documentation

5.100.2.1 WaitUntilUpToSpeedCommand()

```
WaitUntilUpToSpeedCommand::WaitUntilUpToSpeedCommand (
    Flywheel & flywheel,
    int threshold_rpm )
```

Create a [WaitUntilUpToSpeedCommand](#)

Parameters

<code>flywheel</code>	the flywheel system we are commanding
<code>threshold_rpm</code>	the threshold over and under the flywheel target RPM that we define to be acceptable

5.100.3 Member Function Documentation

5.100.3.1 run()

```
bool WaitUntilUpToSpeedCommand::run ( ) [override], [virtual]
```

Run spin_manual Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command_structure/flywheel_commands.h
- src/utils/command_structure/flywheel_commands.cpp

5.101 screen::WidgetConfig Struct Reference

Public Types

- enum **Type** {
 Col , **Row** , **Slider** , **Button** ,
 Checkbox , **Label** , **Text** , **Graph** }

Public Attributes

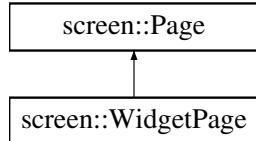
- Type **type**
- union {
 std::vector< [SizedWidget](#) > **widgets**
 [SliderConfig](#) **slider**
 [ButtonConfig](#) **button**
 [CheckboxConfig](#) **checkbox**
 [LabelConfig](#) **label**
 [TextConfig](#) **text**
 [GraphDrawer](#) * **graph**
} **config**

The documentation for this struct was generated from the following file:

- include/subsystems/screen.h

5.102 screen::WidgetPage Class Reference

Inheritance diagram for screen::WidgetPage:



Public Member Functions

- **WidgetPage** ([WidgetConfig &cfg](#))
- void [update](#) (bool was_pressed, int x, int y) override
collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this [Page](#) (only drawn page gets touch updates))
- void [draw](#) (vex::brain::lcd &, bool first_draw, unsigned int frame_number) override
draw stored data to the screen (runs at 10 hz and only runs if this page is in front)

5.102.1 Member Function Documentation

5.102.1.1 draw()

```
void screen::WidgetPage::draw (
    vex::brain::lcd & screen,
    bool first_draw,
    unsigned int frame_number ) [inline], [override], [virtual]
```

draw stored data to the screen (runs at 10 hz and only runs if this page is in front)

Parameters

<i>first_draw</i>	true if we just switched to this page
<i>frame_number</i>	frame of drawing we are on (basically an animation tick)

Reimplemented from [screen::Page](#).

5.102.1.2 update()

```
void screen::WidgetPage::update (
    bool was_pressed,
    int x,
    int y ) [override], [virtual]
```

collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this [Page](#) (only drawn page gets touch updates))

Parameters

<i>was_pressed</i>	true if the screen has been pressed
<i>x</i>	x position of screen press (if the screen was pressed)
<i>y</i>	y position of screen press (if the screen was pressed)

Reimplemented from [screen::Page](#).

The documentation for this class was generated from the following file:

- include/subsystems/screen.h

Chapter 6

File Documentation

6.1 robot_specs.h

```
00001 #pragma once
00002 #include "../core/include/utils/controls/pid.h"
00003 #include "../core/include/utils/controls/feedback_base.h"
00004
00011 typedef struct
00012 {
00013     double robot_radius;
00014
00015     double odom_wheel_diam;
00016     double odom_gear_ratio;
00017     double dist_between_wheels;
00018
00019     double drive_correction_cutoff;
00020
00021     Feedback *drive_feedback;
00022     Feedback *turn_feedback;
00023     PID::pid_config_t correction_pid;
00024
00025 } robot_specs_t;
```

6.2 custom_encoder.h

```
00001 #pragma once
00002 #include "vex.h"
00003
00008 class CustomEncoder : public vex::encoder
00009 {
00010     typedef vex::encoder super;
00011
00012     public:
00018     CustomEncoder(vex::triport::port &port, double ticks_per_rev);
00019
00025     void setRotation(double val, vex::rotationUnits units);
00026
00032     void setPosition(double val, vex::rotationUnits units);
00033
00039     double rotation(vex::rotationUnits units);
00040
00046     double position(vex::rotationUnits units);
00047
00053     double velocity(vex::velocityUnits units);
00054
00055
00056     private:
00057     double tick_scalar;
00058 },
```

6.3 flywheel.h

```

00001 #pragma once
00002
00003 #include "../core/include/utils/controls/feedforward.h"
00004 #include "vex.h"
00005 #include "../core/include/robot_specs.h"
00006 #include "../core/include/utils/controls/pid.h"
00007 #include "../core/include/utils/command_structure/auto_command.h"
00008 #include "../core/include/subsystems/screen.h"
00009 #include <atomic>
00010
00018 class Flywheel
00019 {
00020
00021 public:
00022 // CONSTRUCTORS, GETTERS, AND SETTERS
00031 Flywheel(vex::motor_group &motors, Feedback &feedback, FeedForward &helper, const double ratio,
00032 Filter &filt);
00033
00037 double get_target() const;
00038
00042 double getRPM() const;
00043
00047 vex::motor_group &get_motors() const;
00048
00055 void spin_manual(double speed, directionType dir = fwd);
00056
00062 void spin_rpm(double rpm);
00063
00067 void stop();
00068
00073 bool is_on_target()
00074 {
00075     return fb.is_on_target();
00076 }
00077
00082 screen::Page *Page() const;
00083
00089 AutoCommand *SpinRpmCmd(int rpm)
00090 {
00091
00092     return new FunctionCommand([this, rpm]()
00093             {spin_rpm(rpm); return true; });
00094 }
00095
00100 AutoCommand *WaitUntilUpToSpeedCmd()
00101 {
00102     return new WaitUntilCondition(
00103         new FunctionCondition([this]()
00104             { return is_on_target(); }));
00105 }
00106
00107 private:
00108     friend class FlywheelPage;
00109     friend int spinRPMTask(void *wheelPointer);
00110
00111 vex::motor_group &motors;
00112 bool task_running = false;
00113 Feedback &fb;
00114 FeedForward &ffd;
00115 vex::mutex fb_mut;
00116 double ratio;
00117 std::atomic<double> target_rpm;
00118 Task rpm_task;
00119 Filter &avger;
00120
00121 // Functions for internal use only
00126 void set_target(double value);
00130 double measure_RPM();
00131
00138 void spin_raw(double speed, directionType dir = fwd);
00139 };

```

6.4 pl_mpeg.h

```

00001 #include "vex.h"
00002 /*
00003 PL_MPEG - MPEG1 Video decoder, MP2 Audio decoder, MPEG-PS demuxer
00004
00005 Dominic Szablewski - https://phoboslab.org
00006
00007

```

```
00008 -- LICENSE: The MIT License(MIT)
00009
00010 Copyright(c) 2019 Dominic Szablewski
00011
00012 Permission is hereby granted, free of charge, to any person obtaining a copy of
00013 this software and associated documentation files(the "Software"), to deal in
00014 the Software without restriction, including without limitation the rights to
00015 use, copy, modify, merge, publish, distribute, sublicense, and / or sell copies
00016 of the Software, and to permit persons to whom the Software is furnished to do
00017 so, subject to the following conditions :
00018 The above copyright notice and this permission notice shall be included in all
00019 copies or substantial portions of the Software.
00020 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00021 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00022 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.IN NO EVENT SHALL THE
00023 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00024 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00025 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00026 SOFTWARE.
00027
00028
00029
00030
00031 -- Synopsis
00032
00033 // Define `PL_MPEG_IMPLEMENTATION` in *one* C/C++ file before including this
00034 // library to create the implementation.
00035
00036 #define PL_MPEG_IMPLEMENTATION
00037 #include "plmpeg.h"
00038
00039 // This function gets called for each decoded video frame
00040 void my_video_callback(plm_t *plm, plm_frame_t *frame, void *user) {
00041     // Do something with frame->y.data, frame->cr.data, frame->cb.data
00042 }
00043
00044 // This function gets called for each decoded audio frame
00045 void my_audio_callback(plm_t *plm, plm_samples_t *frame, void *user) {
00046     // Do something with samples->interleaved
00047 }
00048
00049 // Load a .mpg (MPEG Program Stream) file
00050 plm_t *plm = plm_create_with_filename("some-file.mpg");
00051
00052 // Install the video & audio decode callbacks
00053 plm_set_video_decode_callback(plm, my_video_callback, my_data);
00054 plm_set_audio_decode_callback(plm, my_audio_callback, my_data);
00055
00056
00057 // Decode
00058 do {
00059     plm_decode(plm, time_since_last_call);
00060 } while (!plm_has-ended(plm));
00061
00062 // All done
00063 plm_destroy(plm);
00064
00065
00066
00067 -- Documentation
00068
00069 This library provides several interfaces to load, demux and decode MPEG video
00070 and audio data. A high-level API combines the demuxer, video & audio decoders
00071 in an easy to use wrapper.
00072
00073 Lower-level APIs for accessing the demuxer, video decoder and audio decoder,
00074 as well as providing different data sources are also available.
00075
00076 Interfaces are written in an object oriented style, meaning you create object
00077 instances via various different constructor functions (plm_*create()),
00078 do some work on them and later dispose them via plm_*destroy().
00079
00080 plm_* ..... the high-level interface, combining demuxer and decoders
00081 plm_buffer_* ... the data source used by all interfaces
00082 plm_demux_* ... the MPEG-PS demuxer
00083 plm_video_* ... the MPEG1 Video ("mpeg1") decoder
00084 plm_audio_* ... the MPEG1 Audio Layer II ("mp2") decoder
00085
00086
00087 With the high-level interface you have two options to decode video & audio:
00088
00089 1. Use plm_decode() and just hand over the delta time since the last call.
00090     It will decode everything needed and call your callbacks (specified through
00091     plm_set_{video|audio}_decode_callback()) any number of times.
00092
00093 2. Use plm_decode_video() and plm_decode_audio() to decode exactly one
00094     frame of video or audio data at a time. How you handle the synchronization
```

```

00095     of both streams is up to you.
00096
00097 If you only want to decode video *or* audio through these functions, you should
00098 disable the other stream (plm_set_{video|audio}_enabled(FALSE))
00099
00100 Video data is decoded into a struct with all 3 planes (Y, Cr, Cb) stored in
00101 separate buffers. You can either convert this to RGB on the CPU (slow) via the
00102 plm_frame_to_rgb() function or do it on the GPU with the following matrix:
00103
00104 mat4 bt601 = mat4(
00105     1.16438, 0.00000, 1.59603, -0.87079,
00106     1.16438, -0.39176, -0.81297, 0.52959,
00107     1.16438, 2.01723, 0.00000, -1.08139,
00108     0, 0, 0, 1
00109 );
00110 gl_FragColor = vec4(y, cb, cr, 1.0) * bt601;
00111
00112 Audio data is decoded into a struct with either one single float array with the
00113 samples for the left and right channel interleaved, or if the
00114 PLM_AUDIO_SEPARATE_CHANNELS is defined *before* including this library, into
00115 two separate float arrays - one for each channel.
00116
00117
00118 Data can be supplied to the high level interface, the demuxer and the decoders
00119 in three different ways:
00120
00121 1. Using plm_create_from_filename() or with a file handle with
00122     plm_create_from_file().
00123
00124 2. Using plm_create_with_memory() and supplying a pointer to memory that
00125     contains the whole file.
00126
00127 3. Using plm_create_with_buffer(), supplying your own plm_buffer_t instance and
00128     periodically writing to this buffer.
00129
00130 When using your own plm_buffer_t instance, you can fill this buffer using
00131 plm_buffer_write(). You can either monitor plm_buffer_get_remaining() and push
00132 data when appropriate, or install a callback on the buffer with
00133 plm_buffer_set_load_callback() that gets called whenever the buffer needs more
00134 data.
00135
00136 A buffer created with plm_buffer_create_with_capacity() is treated as a ring
00137 buffer, meaning that data that has already been read, will be discarded. In
00138 contrast, a buffer created with plm_buffer_create_for_appending() will keep all
00139 data written to it in memory. This enables seeking in the already loaded data.
00140
00141
00142 There should be no need to use the lower level plm_demux_*, plm_video_* and
00143 plm_audio_* functions, if all you want to do is read/decode an MPEG-PS file.
00144 However, if you get raw mpeg1video data or raw mp2 audio data from a different
00145 source, these functions can be used to decode the raw data directly. Similarly,
00146 if you only want to analyze an MPEG-PS file or extract raw video or audio
00147 packets from it, you can use the plm_demux_* functions.
00148
00149
00150 This library uses malloc(), realloc() and free() to manage memory. Typically
00151 all allocation happens up-front when creating the interface. However, the
00152 default buffer size may be too small for certain inputs. In these cases plmpeg
00153 will realloc() the buffer with a larger size whenever needed. You can configure
00154 the default buffer size by defining PLM_BUFFER_DEFAULT_SIZE *before*
00155 including this library.
00156
00157 You can also define PLM_MALLOC, PLM_REALLOC and PLM_FREE to provide your own
00158 memory management functions.
00159
00160
00161 See below for detailed the API documentation.
00162
00163 */
00164
00165
00166 #ifndef PL_MPEG_H
00167 #define PL_MPEG_H
00168
00169 #include <stdint.h>
00170 // #include <stdio.h>
00171
00172
00173 #ifdef __cplusplus
00174 extern "C" {
00175 #endif
00176
00177 // -----
00178 // Public Data Types
00179
00180
00181 // Object types for the various interfaces

```

```

00182
00183 typedef struct plm_t plm_t;
00184 typedef struct plm_buffer_t plm_buffer_t;
00185 typedef struct plm_demux_t plm_demux_t;
00186 typedef struct plm_video_t plm_video_t;
00187 typedef struct plm_audio_t plm_audio_t;
00188
00189
00190 // Demuxed MPEG PS packet
00191 // The type maps directly to the various MPEG-PES start codes. PTS is the
00192 // presentation time stamp of the packet in seconds. Note that not all packets
00193 // have a PTS value, indicated by PLM_PACKET_INVALID_TS.
00194
00195 #define PLM_PACKET_INVALID_TS -1
00196
00197 typedef struct {
00198     int type;
00199     double pts;
00200     size_t length;
00201     uint8_t *data;
00202 } plm_packet_t;
00203
00204
00205 // Decoded Video Plane
00206 // The byte length of the data is width * height. Note that different planes
00207 // have different sizes: the Luma plane (Y) is double the size of each of
00208 // the two Chroma planes (Cr, Cb) - i.e. 4 times the byte length.
00209 // Also note that the size of the plane does *not* denote the size of the
00210 // displayed frame. The sizes of planes are always rounded up to the nearest
00211 // macroblock (16px).
00212
00213 typedef struct {
00214     unsigned int width;
00215     unsigned int height;
00216     uint8_t *data;
00217 } plm_plane_t;
00218
00219
00220 // Decoded Video Frame
00221 // width and height denote the desired display size of the frame. This may be
00222 // different from the internal size of the 3 planes.
00223
00224 typedef struct {
00225     double time;
00226     unsigned int width;
00227     unsigned int height;
00228     plm_plane_t y;
00229     plm_plane_t cr;
00230     plm_plane_t cb;
00231 } plm_frame_t;
00232
00233
00234 // Callback function type for decoded video frames used by the high-level
00235 // plm_* interface
00236
00237 typedef void(*plm_video_decode_callback)
00238     (plm_t *self, plm_frame_t *frame, void *user);
00239
00240
00241 // Decoded Audio Samples
00242 // Samples are stored as normalized (-1, 1) float either interleaved, or if
00243 // PLM_AUDIO_SEPARATE_CHANNELS is defined, in two separate arrays.
00244 // The `count` is always PLM_AUDIO_SAMPLES_PER_FRAME and just there for
00245 // convenience.
00246
00247 #define PLM_AUDIO_SAMPLES_PER_FRAME 1152
00248
00249 typedef struct {
00250     double time;
00251     unsigned int count;
00252     #ifdef PLM_AUDIO_SEPARATE_CHANNELS
00253         float left[PLM_AUDIO_SAMPLES_PER_FRAME];
00254         float right[PLM_AUDIO_SAMPLES_PER_FRAME];
00255     #else
00256         float interleaved[PLM_AUDIO_SAMPLES_PER_FRAME * 2];
00257     #endif
00258 } plm_samples_t;
00259
00260
00261 // Callback function type for decoded audio samples used by the high-level
00262 // plm_* interface
00263
00264 typedef void(*plm_audio_decode_callback)
00265     (plm_t *self, plm_samples_t *samples, void *user);
00266
00267
00268 // Callback function for plm_buffer when it needs more data

```

```
00269
00270 typedef void(*plm_buffer_load_callback)(plm_buffer_t *self, void *user);
00271
00272
00273
00274 // -----
00275 // plm_* public API
00276 // High-Level API for loading/demuxing/decoding MPEG-PS data
00277
00278
00279 // Create a plmpeg instance with a filename. Returns NULL if the file could not
00280 // be opened.
00281
00282 plm_t *plm_create_with_filename(const char *filename);
00283
00284
00285 // Create a plmpeg instance with a file handle. Pass TRUE to close_when_done to
00286 // let plmpeg call fclose() on the handle when plm_destroy() is called.
00287
00288 plm_t *plm_create_with_file(FIL *fh, int close_when_done);
00289
00290
00291 // Create a plmpeg instance with a pointer to memory as source. This assumes the
00292 // whole file is in memory. The memory is not copied. Pass TRUE to
00293 // free_when_done to let plmpeg call free() on the pointer when plm_destroy()
00294 // is called.
00295
00296 plm_t *plm_create_with_memory(uint8_t *bytes, size_t length, int free_when_done);
00297
00298
00299 // Create a plmpeg instance with a plm_buffer as source. Pass TRUE to
00300 // destroy_when_done to let plmpeg call plm_buffer_destroy() on the buffer when
00301 // plm_destroy() is called.
00302
00303 plm_t *plm_create_with_buffer(plm_buffer_t *buffer, int destroy_when_done);
00304
00305
00306 // Destroy a plmpeg instance and free all data.
00307
00308 void plm_destroy(plm_t *self);
00309
00310
00311 // Get whether we have headers on all available streams and we can accurately
00312 // report the number of video/audio streams, video dimensions, framerate and
00313 // audio samplerate.
00314 // This returns FALSE if the file is not an MPEG-PS file or - when not using a
00315 // file as source - when not enough data is available yet.
00316
00317 int plm_has_headers(plm_t *self);
00318
00319
00320 // Get or set whether video decoding is enabled. Default TRUE.
00321
00322 int plm_get_video_enabled(plm_t *self);
00323 void plm_set_video_enabled(plm_t *self, int enabled);
00324
00325
00326 // Get the number of video streams (0--1) reported in the system header.
00327
00328 int plm_get_num_video_streams(plm_t *self);
00329
00330
00331 // Get the display width/height of the video stream.
00332
00333 int plm_get_width(plm_t *self);
00334 int plm_get_height(plm_t *self);
00335
00336
00337 // Get the framerate of the video stream in frames per second.
00338
00339 double plm_get_framerate(plm_t *self);
00340
00341
00342 // Get or set whether audio decoding is enabled. Default TRUE.
00343
00344 int plm_get_audio_enabled(plm_t *self);
00345 void plm_set_audio_enabled(plm_t *self, int enabled);
00346
00347
00348 // Get the number of audio streams (0--4) reported in the system header.
00349
00350 int plm_get_num_audio_streams(plm_t *self);
00351
00352
00353 // Set the desired audio stream (0--3). Default 0.
00354
00355 void plm_set_audio_stream(plm_t *self, int stream_index);
```

```

00356
00357
00358 // Get the samplerate of the audio stream in samples per second.
00359
00360 int plm_get_samplerate(plm_t *self);
00361
00362
00363 // Get or set the audio lead time in seconds - the time in which audio samples
00364 // are decoded in advance (or behind) the video decode time. Typically this
00365 // should be set to the duration of the buffer of the audio API that you use
00366 // for output. E.g. for SDL2: (SDL_AudioSpec.samples / samplerate)
00367
00368 double plm_get_audio_lead_time(plm_t *self);
00369 void plm_set_audio_lead_time(plm_t *self, double lead_time);
00370
00371
00372 // Get the current internal time in seconds.
00373
00374 double plm_get_time(plm_t *self);
00375
00376
00377 // Get the video duration of the underlying source in seconds.
00378
00379 double plm_get_duration(plm_t *self);
00380
00381
00382 // Rewind all buffers back to the beginning.
00383
00384 void plm_rewind(plm_t *self);
00385
00386
00387 // Get or set looping. Default FALSE.
00388
00389 int plm_get_loop(plm_t *self);
00390 void plm_set_loop(plm_t *self, int loop);
00391
00392
00393 // Get whether the file has ended. If looping is enabled, this will always
00394 // return FALSE.
00395
00396 int plm_hasEnded(plm_t *self);
00397
00398
00399 // Set the callback for decoded video frames used with plm_decode(). If no
00400 // callback is set, video data will be ignored and not be decoded. The *user
00401 // Parameter will be passed to your callback.
00402
00403 void plm_set_video_decode_callback(plm_t *self, plm_video_decode_callback fp, void *user);
00404
00405
00406 // Set the callback for decoded audio samples used with plm_decode(). If no
00407 // callback is set, audio data will be ignored and not be decoded. The *user
00408 // Parameter will be passed to your callback.
00409
00410 void plm_set_audio_decode_callback(plm_t *self, plm_audio_decode_callback fp, void *user);
00411
00412
00413 // Advance the internal timer by seconds and decode video/audio up to this time.
00414 // This will call the video_decode_callback and audio_decode_callback any number
00415 // of times. A frame-skip is not implemented, i.e. everything up to current time
00416 // will be decoded.
00417
00418 void plm_decode(plm_t *self, double seconds);
00419
00420
00421 // Decode and return one video frame. Returns NULL if no frame could be decoded
00422 // (either because the source ended or data is corrupt). If you only want to
00423 // decode video, you should disable audio via plm_set_audio_enabled().
00424 // The returned plm_frame_t is valid until the next call to plm_decode_video()
00425 // or until plm_destroy() is called.
00426
00427 plm_frame_t *plm_decode_video(plm_t *self);
00428
00429
00430 // Decode and return one audio frame. Returns NULL if no frame could be decoded
00431 // (either because the source ended or data is corrupt). If you only want to
00432 // decode audio, you should disable video via plm_set_video_enabled().
00433 // The returned plm_samples_t is valid until the next call to plm_decode_audio()
00434 // or until plm_destroy() is called.
00435
00436 plm_samples_t *plm_decode_audio(plm_t *self);
00437
00438
00439 // Seek to the specified time, clamped between 0 -- duration. This can only be
00440 // used when the underlying plm_buffer is seekable, i.e. for files, fixed
00441 // memory buffers or _forAppending buffers.
00442 // If seek_exact is TRUE this will seek to the exact time, otherwise it will

```

```

00443 // seek to the last intra frame just before the desired time. Exact seeking can
00444 // be slow, because all frames up to the sought one have to be decoded on top of
00445 // the previous intra frame.
00446 // If seeking succeeds, this function will call the video_decode_callback
00447 // exactly once with the target frame. If audio is enabled, it will also call
00448 // the audio_decode_callback any number of times, until the audio_lead_time is
00449 // satisfied.
00450 // Returns TRUE if seeking succeeded or FALSE if no frame could be found.
00451
00452 int plm_seek(plm_t *self, double time, int seek_exact);
00453
00454
00455 // Similar to plm_seek(), but will not call the video_decode_callback,
00456 // audio_decode_callback or make any attempts to sync audio.
00457 // Returns the found frame or NULL if no frame could be found.
00458
00459 plm_frame_t *plm_seek_frame(plm_t *self, double time, int seek_exact);
00460
00461
00462
00463 // -----
00464 // plm_buffer public API
00465 // Provides the data source for all other plm_* interfaces
00466
00467
00468 // The default size for buffers created from files or by the high-level API
00469
00470 #ifndef PLM_BUFFER_DEFAULT_SIZE
00471 #define PLM_BUFFER_DEFAULT_SIZE (128 * 1024)
00472 #endif
00473
00474
00475 // Create a buffer instance with a filename. Returns NULL if the file could not
00476 // be opened.
00477
00478 plm_buffer_t *plm_buffer_create_with_filename(const char *filename);
00479
00480
00481 // Create a buffer instance with a file handle. Pass TRUE to close_when_done
00482 // to let plmpeg call fclose() on the handle when plm_destroy() is called.
00483
00484 plm_buffer_t *plm_buffer_create_with_file(FIL *fh, int close_when_done);
00485
00486
00487 // Create a buffer instance with a pointer to memory as source. This assumes
00488 // the whole file is in memory. The bytes are not copied. Pass 1 to
00489 // free_when_done to let plmpeg call free() on the pointer when plm_destroy()
00490 // is called.
00491
00492 plm_buffer_t *plm_buffer_create_with_memory(uint8_t *bytes, size_t length, int free_when_done);
00493
00494
00495 // Create an empty buffer with an initial capacity. The buffer will grow
00496 // as needed. Data that has already been read, will be discarded.
00497
00498 plm_buffer_t *plm_buffer_create_with_capacity(size_t capacity);
00499
00500
00501 // Create an empty buffer with an initial capacity. The buffer will grow
00502 // as needed. Decoded data will *not* be discarded. This can be used when
00503 // loading a file over the network, without needing to throttle the download.
00504 // It also allows for seeking in the already loaded data.
00505
00506 plm_buffer_t *plm_buffer_create_for_appending(size_t initial_capacity);
00507
00508
00509 // Destroy a buffer instance and free all data
00510
00511 void plm_buffer_destroy(plm_buffer_t *self);
00512
00513
00514 // Copy data into the buffer. If the data to be written is larger than the
00515 // available space, the buffer will realloc() with a larger capacity.
00516 // Returns the number of bytes written. This will always be the same as the
00517 // passed in length, except when the buffer was created _with_memory() for
00518 // which _write() is forbidden.
00519
00520 size_t plm_buffer_write(plm_buffer_t *self, uint8_t *bytes, size_t length);
00521
00522
00523 // Mark the current byte length as the end of this buffer and signal that no
00524 // more data is expected to be written to it. This function should be called
00525 // just after the last plm_buffer_write().
00526 // For _with_capacity buffers, this is cleared on a plm_buffer_rewind().
00527
00528 void plm_buffer_signal_end(plm_buffer_t *self);
00529

```

```
00530
00531 // Set a callback that is called whenever the buffer needs more data
00532
00533 void plm_buffer_set_load_callback(plm_buffer_t *self, plm_buffer_load_callback fp, void *user);
00534
00535
00536 // Rewind the buffer back to the beginning. When loading from a file handle,
00537 // this also seeks to the beginning of the file.
00538
00539 void plm_buffer_rewind(plm_buffer_t *self);
00540
00541
00542 // Get the total size. For files, this returns the file size. For all other
00543 // types it returns the number of bytes currently in the buffer.
00544
00545 size_t plm_buffer_get_size(plm_buffer_t *self);
00546
00547
00548 // Get the number of remaining (yet unread) bytes in the buffer. This can be
00549 // useful to throttle writing.
00550
00551 size_t plm_buffer_get_remaining(plm_buffer_t *self);
00552
00553
00554 // Get whether the read position of the buffer is at the end and no more data
00555 // is expected.
00556
00557 int plm_buffer_has-ended(plm_buffer_t *self);
00558
00559
00560
00561 // -----
00562 // plm_demux public API
00563 // Demux an MPEG Program Stream (PS) data into separate packages
00564
00565
00566 // Various Packet Types
00567
00568 static const int PLM_DEMUX_PACKET_PRIVATE = 0xBD;
00569 static const int PLM_DEMUX_PACKET_AUDIO_1 = 0xC0;
00570 static const int PLM_DEMUX_PACKET_AUDIO_2 = 0xC1;
00571 static const int PLM_DEMUX_PACKET_AUDIO_3 = 0xC2;
00572 static const int PLM_DEMUX_PACKET_AUDIO_4 = 0xC3;
00573 static const int PLM_DEMUX_PACKET_VIDEO_1 = 0xE0;
00574
00575
00576 // Create a demuxer with a plm_buffer as source. This will also attempt to read
00577 // the pack and system headers from the buffer.
00578
00579 plm_demux_t *plm_demux_create(plm_buffer_t *buffer, int destroy_when_done);
00580
00581
00582 // Destroy a demuxer and free all data.
00583
00584 void plm_demux_destroy(plm_demux_t *self);
00585
00586
00587 // Returns TRUE/FALSE whether pack and system headers have been found. This will
00588 // attempt to read the headers if non are present yet.
00589
00590 int plm_demux_has_headers(plm_demux_t *self);
00591
00592
00593 // Returns the number of video streams found in the system header. This will
00594 // attempt to read the system header if non is present yet.
00595
00596 int plm_demux_get_num_video_streams(plm_demux_t *self);
00597
00598
00599 // Returns the number of audio streams found in the system header. This will
00600 // attempt to read the system header if non is present yet.
00601
00602 int plm_demux_get_num_audio_streams(plm_demux_t *self);
00603
00604
00605 // Rewind the internal buffer. See plm_buffer_rewind().
00606
00607 void plm_demux_rewind(plm_demux_t *self);
00608
00609
00610 // Get whether the file has ended. This will be cleared on seeking or rewind.
00611
00612 int plm_demux_has-ended(plm_demux_t *self);
00613
00614
00615 // Seek to a packet of the specified type with a PTS just before specified time.
00616 // If force_intra is TRUE, only packets containing an intra frame will be
```

```
00617 // considered - this only makes sense when the type is PLM_DEMUX_PACKET_VIDEO_1.
00618 // Note that the specified time is considered 0-based, regardless of the first
00619 // PTS in the data source.
00620
00621 plm_packet_t *plm_demux_seek(plm_demux_t *self, double time, int type, int force_intra);
00622
00623
00624 // Get the PTS of the first packet of this type. Returns PLM_PACKET_INVALID_TS
00625 // if not packet of this packet type can be found.
00626
00627 double plm_demux_get_start_time(plm_demux_t *self, int type);
00628
00629
00630 // Get the duration for the specified packet type - i.e. the span between the
00631 // the first PTS and the last PTS in the data source. This only makes sense when
00632 // the underlying data source is a file or fixed memory.
00633
00634 double plm_demux_get_duration(plm_demux_t *self, int type);
00635
00636
00637 // Decode and return the next packet. The returned packet_t is valid until
00638 // the next call to plm_demux_decode() or until the demuxer is destroyed.
00639
00640 plm_packet_t *plm_demux_decode(plm_demux_t *self);
00641
00642
00643
00644 // -----
00645 // plm_video public API
00646 // Decode MPEG1 Video ("mpeg1") data into raw YCrCb frames
00647
00648
00649 // Create a video decoder with a plm_buffer as source.
00650
00651 plm_video_t *plm_video_create_with_buffer(plm_buffer_t *buffer, int destroy_when_done);
00652
00653
00654 // Destroy a video decoder and free all data.
00655
00656 void plm_video_destroy(plm_video_t *self);
00657
00658
00659 // Get whether a sequence header was found and we can accurately report on
00660 // dimensions and framerate.
00661
00662 int plm_video_has_header(plm_video_t *self);
00663
00664
00665 // Get the framerate in frames per second.
00666
00667 double plm_video_get_framerate(plm_video_t *self);
00668
00669
00670 // Get the display width/height.
00671
00672 int plm_video_get_width(plm_video_t *self);
00673 int plm_video_get_height(plm_video_t *self);
00674
00675
00676 // Set "no delay" mode. When enabled, the decoder assumes that the video does
00677 // *not* contain any B-Frames. This is useful for reducing lag when streaming.
00678 // The default is FALSE.
00679
00680 void plm_video_set_no_delay(plm_video_t *self, int no_delay);
00681
00682
00683 // Get the current internal time in seconds.
00684
00685 double plm_video_get_time(plm_video_t *self);
00686
00687
00688 // Set the current internal time in seconds. This is only useful when you
00689 // manipulate the underlying video buffer and want to enforce a correct
00690 // timestamps.
00691
00692 void plm_video_set_time(plm_video_t *self, double time);
00693
00694
00695 // Rewind the internal buffer. See plm_buffer_rewind().
00696
00697 void plm_video_rewind(plm_video_t *self);
00698
00699
00700 // Get whether the file has ended. This will be cleared on rewind.
00701
00702 int plm_video_hasEnded(plm_video_t *self);
00703
```

```
00704
00705 // Decode and return one frame of video and advance the internal time by
00706 // 1/framerate seconds. The returned frame_t is valid until the next call of
00707 // plm_video_decode() or until the video decoder is destroyed.
00708
00709 plm_frame_t *plm_video_decode(plm_video_t *self);
00710
00711
00712 // Convert the YCrCb data of a frame into interleaved R G B data. The stride
00713 // specifies the width in bytes of the destination buffer. I.e. the number of
00714 // bytes from one line to the next. The stride must be at least
00715 // (frame->width * bytes_per_pixel). The buffer pointed to by *dest must have a
00716 // size of at least (stride * frame->height).
00717 // Note that the alpha component of the dest buffer is always left untouched.
00718
00719 void plm_frame_to_rgb(plm_frame_t *frame, uint8_t *dest, int stride);
00720 void plm_frame_to_bgr(plm_frame_t *frame, uint8_t *dest, int stride);
00721 void plm_frame_to_rgba(plm_frame_t *frame, uint8_t *dest, int stride);
00722 void plm_frame_to_bgra(plm_frame_t *frame, uint8_t *dest, int stride);
00723 void plm_frame_to_argb(plm_frame_t *frame, uint8_t *dest, int stride);
00724 void plm_frame_to_abgr(plm_frame_t *frame, uint8_t *dest, int stride);
00725
00726
00727 // -----
00728 // plm_audio public API
00729 // Decode MPEG-1 Audio Layer II ("mp2") data into raw samples
00730
00731
00732 // Create an audio decoder with a plm_buffer as source.
00733
00734 plm_audio_t *plm_audio_create_with_buffer(plm_buffer_t *buffer, int destroy_when_done);
00735
00736
00737 // Destroy an audio decoder and free all data.
00738
00739 void plm_audio_destroy(plm_audio_t *self);
00740
00741
00742 // Get whether a frame header was found and we can accurately report on
00743 // samplerate.
00744
00745 int plm_audio_has_header(plm_audio_t *self);
00746
00747
00748 // Get the samplerate in samples per second.
00749
00750 int plm_audio_get_samplerate(plm_audio_t *self);
00751
00752
00753 // Get the current internal time in seconds.
00754
00755 double plm_audio_get_time(plm_audio_t *self);
00756
00757
00758 // Set the current internal time in seconds. This is only useful when you
00759 // manipulate the underlying video buffer and want to enforce a correct
00760 // timestamps.
00761
00762 void plm_audio_set_time(plm_audio_t *self, double time);
00763
00764
00765 // Rewind the internal buffer. See plm_buffer_rewind().
00766
00767 void plm_audio_rewind(plm_audio_t *self);
00768
00769
00770 // Get whether the file has ended. This will be cleared on rewind.
00771
00772 int plm_audio_hasEnded(plm_audio_t *self);
00773
00774
00775 // Decode and return one "frame" of audio and advance the internal time by
00776 // (PLM_AUDIO_SAMPLES_PER_FRAME/samplerate) seconds. The returned samples_t
00777 // is valid until the next call of plm_audio_decode() or until the audio
00778 // decoder is destroyed.
00779
00780 plm_samples_t *plm_audio_decode(plm_audio_t *self);
00781
00782
00783
00784 #ifdef __cplusplus
00785 }
00786 #endif
00787
00788 #endif // PL_MPEG_H
00789
00790
```

```
00791
00792
00793
00794 // -----
00795 // -----
00796 // IMPLEMENTATION
00797
00798 #ifdef PL_MPEG_IMPLEMENTATION
00799
00800 #include <string.h>
00801 #include <stdlib.h>
00802
00803 #ifndef TRUE
00804 #define TRUE 1
00805 #define FALSE 0
00806 #endif
00807
00808 #ifndef PLM_MALLOC
00809     #define PLM_MALLOC(sz) malloc(sz)
00810     #define PLM_FREE(p) free(p)
00811     #define PLM_REALLOC(p, sz) realloc(p, sz)
00812 #endif
00813
00814 #define PLM_UNUSED(expr) (void)(expr)
00815
00816
00817 // -----
00818 // plm (high-level interface) implementation
00819
00820 struct plm_t {
00821     plm_demux_t *demux;
00822     double time;
00823     int has-ended;
00824     int loop;
00825     int has_decoders;
00826
00827     int video_enabled;
00828     int video_packet_type;
00829     plm_buffer_t *video_buffer;
00830     plm_video_t *video_decoder;
00831
00832     int audio_enabled;
00833     int audio_stream_index;
00834     int audio_packet_type;
00835     double audio_lead_time;
00836     plm_buffer_t *audio_buffer;
00837     plm_audio_t *audio_decoder;
00838
00839     plm_video_decode_callback video_decode_callback;
00840     void *video_decode_callback_user_data;
00841
00842     plm_audio_decode_callback audio_decode_callback;
00843     void *audio_decode_callback_user_data;
00844 };
00845
00846 int plm_init_decoders(plm_t *self);
00847 void plm_handle_end(plm_t *self);
00848 void plm_read_video_packet(plm_buffer_t *buffer, void *user);
00849 void plm_read_audio_packet(plm_buffer_t *buffer, void *user);
00850 void plm_read_packets(plm_t *self, int requested_type);
00851
00852 plm_t *plm_create_with_filename(const char *filename) {
00853     plm_buffer_t *buffer = plm_buffer_create_with_filename(filename);
00854     if (!buffer) {
00855         return NULL;
00856     }
00857     return plm_create_with_buffer(buffer, TRUE);
00858 }
00859
00860 plm_t *plm_create_with_file(FIL *fh, int close_when_done) {
00861     plm_buffer_t *buffer = plm_buffer_create_with_file(fh, close_when_done);
00862     return plm_create_with_buffer(buffer, TRUE);
00863 }
00864
00865 plm_t *plm_create_with_memory(uint8_t *bytes, size_t length, int free_when_done) {
00866     plm_buffer_t *buffer = plm_buffer_create_with_memory(bytes, length, free_when_done);
00867     return plm_create_with_buffer(buffer, TRUE);
00868 }
00869
00870 plm_t *plm_create_with_buffer(plm_buffer_t *buffer, int destroy_when_done) {
00871     plm_t *self = (plm_t *)PLM_MALLOC(sizeof(plm_t));
00872     memset(self, 0, sizeof(plm_t));
00873
00874     self->demux = plm_demux_create(buffer, destroy_when_done);
00875     self->video_enabled = TRUE;
00876     self->audio_enabled = TRUE;
00877     plm_init_decoders(self);
```

```

00878
00879     return self;
00880 }
00881
00882 int plm_init_decoders(plm_t *self) {
00883     if (self->has_decoders) {
00884         return TRUE;
00885     }
00886
00887     if (!plm_demux_has_headers(self->demux)) {
00888         return FALSE;
00889     }
00890
00891     if (plm_demux_get_num_video_streams(self->demux) > 0) {
00892         if (self->video_enabled) {
00893             self->video_packet_type = PLM_DEMUX_PACKET_VIDEO_1;
00894         }
00895         self->video_buffer = plm_buffer_create_with_capacity(PLM_BUFFER_DEFAULT_SIZE);
00896         plm_buffer_set_load_callback(self->video_buffer, plm_read_video_packet, self);
00897     }
00898
00899     if (plm_demux_get_num_audio_streams(self->demux) > 0) {
00900         if (self->audio_enabled) {
00901             self->audio_packet_type = PLM_DEMUX_PACKET_AUDIO_1 + self->audio_stream_index;
00902         }
00903         self->audio_buffer = plm_buffer_create_with_capacity(PLM_BUFFER_DEFAULT_SIZE);
00904         plm_buffer_set_load_callback(self->audio_buffer, plm_read_audio_packet, self);
00905     }
00906
00907     if (self->video_buffer) {
00908         self->video_decoder = plm_video_create_with_buffer(self->video_buffer, TRUE);
00909     }
00910
00911     if (self->audio_buffer) {
00912         self->audio_decoder = plm_audio_create_with_buffer(self->audio_buffer, TRUE);
00913     }
00914
00915     self->has_decoders = TRUE;
00916     return TRUE;
00917 }
00918
00919 void plm_destroy(plm_t *self) {
00920     if (self->video_decoder) {
00921         plm_video_destroy(self->video_decoder);
00922     }
00923     if (self->audio_decoder) {
00924         plm_audio_destroy(self->audio_decoder);
00925     }
00926
00927     plm_demux_destroy(self->demux);
00928     PLM_FREE(self);
00929 }
00930
00931 int plm_get_audio_enabled(plm_t *self) {
00932     return self->audio_enabled;
00933 }
00934
00935 int plm_has_headers(plm_t *self) {
00936     if (!plm_demux_has_headers(self->demux)) {
00937         return FALSE;
00938     }
00939
00940     if (!plm_init_decoders(self)) {
00941         return FALSE;
00942     }
00943
00944     if (
00945         (self->video_decoder && !plm_video_has_header(self->video_decoder)) ||
00946         (self->audio_decoder && !plm_audio_has_header(self->audio_decoder))
00947     ) {
00948         return FALSE;
00949     }
00950
00951     return TRUE;
00952 }
00953
00954 void plm_set_audio_enabled(plm_t *self, int enabled) {
00955     self->audio_enabled = enabled;
00956
00957     if (!enabled) {
00958         self->audio_packet_type = 0;
00959         return;
00960     }
00961
00962     self->audio_packet_type = (plm_init_decoders(self) && self->audio_decoder)
00963         ? PLM_DEMUX_PACKET_AUDIO_1 + self->audio_stream_index
00964         : 0;

```

```

00965 }
00966
00967 void plm_set_audio_stream(plm_t *self, int stream_index) {
00968     if (stream_index < 0 || stream_index > 3) {
00969         return;
00970     }
00971     self->audio_stream_index = stream_index;
00972
00973     // Set the correct audio_packet_type
00974     plm_set_audio_enabled(self, self->audio_enabled);
00975 }
00976
00977 int plm_get_video_enabled(plm_t *self) {
00978     return self->video_enabled;
00979 }
00980
00981 void plm_set_video_enabled(plm_t *self, int enabled) {
00982     self->video_enabled = enabled;
00983
00984     if (!enabled) {
00985         self->video_packet_type = 0;
00986         return;
00987     }
00988
00989     self->video_packet_type = (plm_init_decoders(self) && self->video_decoder)
00990         ? PLM_DEMUX_PACKET_VIDEO_1
00991         : 0;
00992 }
00993
00994 int plm_get_num_video_streams(plm_t *self) {
00995     return plm_demux_get_num_video_streams(self->demux);
00996 }
00997
00998 int plm_get_width(plm_t *self) {
00999     return (plm_init_decoders(self) && self->video_decoder)
01000         ? plm_video_get_width(self->video_decoder)
01001         : 0;
01002 }
01003
01004 int plm_get_height(plm_t *self) {
01005     return (plm_init_decoders(self) && self->video_decoder)
01006         ? plm_video_get_height(self->video_decoder)
01007         : 0;
01008 }
01009
01010 double plm_get_framerate(plm_t *self) {
01011     return (plm_init_decoders(self) && self->video_decoder)
01012         ? plm_video_get_framerate(self->video_decoder)
01013         : 0;
01014 }
01015
01016 int plm_get_num_audio_streams(plm_t *self) {
01017     return plm_demux_get_num_audio_streams(self->demux);
01018 }
01019
01020 int plm_get_samplerate(plm_t *self) {
01021     return (plm_init_decoders(self) && self->audio_decoder)
01022         ? plm_audio_get_samplerate(self->audio_decoder)
01023         : 0;
01024 }
01025
01026 double plm_get_audio_lead_time(plm_t *self) {
01027     return self->audio_lead_time;
01028 }
01029
01030 void plm_set_audio_lead_time(plm_t *self, double lead_time) {
01031     self->audio_lead_time = lead_time;
01032 }
01033
01034 double plm_get_time(plm_t *self) {
01035     return self->time;
01036 }
01037
01038 double plm_get_duration(plm_t *self) {
01039     return plm_demux_get_duration(self->demux, PLM_DEMUX_PACKET_VIDEO_1);
01040 }
01041
01042 void plm_rewind(plm_t *self) {
01043     if (self->video_decoder) {
01044         plm_video_rewind(self->video_decoder);
01045     }
01046
01047     if (self->audio_decoder) {
01048         plm_audio_rewind(self->audio_decoder);
01049     }
01050
01051     plm_demux_rewind(self->demux);

```

```

01052     self->time = 0;
01053 }
01054
01055 int plm_get_loop(plm_t *self) {
01056     return self->loop;
01057 }
01058
01059 void plm_set_loop(plm_t *self, int loop) {
01060     self->loop = loop;
01061 }
01062
01063 int plm_has-ended(plm_t *self) {
01064     return self->has-ended;
01065 }
01066
01067 void plm_set_video_decode_callback(plm_t *self, plm_video_decode_callback fp, void *user) {
01068     self->video_decode_callback = fp;
01069     self->video_decode_callback_user_data = user;
01070 }
01071
01072 void plm_set_audio_decode_callback(plm_t *self, plm_audio_decode_callback fp, void *user) {
01073     self->audio_decode_callback = fp;
01074     self->audio_decode_callback_user_data = user;
01075 }
01076
01077 void plm_decode(plm_t *self, double tick) {
01078     if (!plm_init_decoders(self)) {
01079         return;
01080     }
01081
01082     int decode_video = (self->video_decode_callback && self->video_packet_type);
01083     int decode_audio = (self->audio_decode_callback && self->audio_packet_type);
01084
01085     if (!decode_video && !decode_audio) {
01086         // Nothing to do here
01087         return;
01088     }
01089
01090     int did_decode = FALSE;
01091     int decode_video_failed = FALSE;
01092     int decode_audio_failed = FALSE;
01093
01094     double video_target_time = self->time + tick;
01095     double audio_target_time = self->time + tick + self->audio_lead_time;
01096
01097     do {
01098         did_decode = FALSE;
01099
01100         if (decode_video && plm_video_get_time(self->video_decoder) < video_target_time) {
01101             plm_frame_t *frame = plm_video_decode(self->video_decoder);
01102             if (frame) {
01103                 self->video_decode_callback(self, frame, self->video_decode_callback_user_data);
01104                 did_decode = TRUE;
01105             }
01106             else {
01107                 decode_video_failed = TRUE;
01108             }
01109         }
01110
01111         if (decode_audio && plm_audio_get_time(self->audio_decoder) < audio_target_time) {
01112             plm_samples_t *samples = plm_audio_decode(self->audio_decoder);
01113             if (samples) {
01114                 self->audio_decode_callback(self, samples, self->audio_decode_callback_user_data);
01115                 did_decode = TRUE;
01116             }
01117             else {
01118                 decode_audio_failed = TRUE;
01119             }
01120         }
01121     } while (did_decode);
01122
01123     // Did all sources we wanted to decode fail and the demuxer is at the end?
01124     if (
01125         (!decode_video || decode_video_failed) &&
01126         (!decode_audio || decode_audio_failed) &&
01127         plm_demux_has-ended(self->demux)
01128     ) {
01129         plm_handle_end(self);
01130         return;
01131     }
01132
01133     self->time += tick;
01134 }
01135
01136 plm_frame_t *plm_decode_video(plm_t *self) {
01137     if (!plm_init_decoders(self)) {
01138         return NULL;

```

```

01139     }
01140
01141     if (!self->video_packet_type) {
01142         return NULL;
01143     }
01144
01145     plm_frame_t *frame = plm_video_decode(self->video_decoder);
01146     if (frame) {
01147         self->time = frame->time;
01148     }
01149     else if (plm_demux_has-ended(self->demux)) {
01150         plm_handle_end(self);
01151     }
01152     return frame;
01153 }
01154
01155 plm_samples_t *plm_decode_audio(plm_t *self) {
01156     if (!plm_init_decoders(self)) {
01157         return NULL;
01158     }
01159
01160     if (!self->audio_packet_type) {
01161         return NULL;
01162     }
01163
01164     plm_samples_t *samples = plm_audio_decode(self->audio_decoder);
01165     if (samples) {
01166         self->time = samples->time;
01167     }
01168     else if (plm_demux_has-ended(self->demux)) {
01169         plm_handle_end(self);
01170     }
01171     return samples;
01172 }
01173
01174 void plm_handle_end(plm_t *self) {
01175     if (self->loop) {
01176         plm_rewind(self);
01177     }
01178     else {
01179         self->has-ended = TRUE;
01180     }
01181 }
01182
01183 void plm_read_video_packet(plm_buffer_t *buffer, void *user) {
01184     PLM_UNUSED(buffer);
01185     plm_t *self = (plm_t *)user;
01186     plm_read_packets(self, self->video_packet_type);
01187 }
01188
01189 void plm_read_audio_packet(plm_buffer_t *buffer, void *user) {
01190     PLM_UNUSED(buffer);
01191     plm_t *self = (plm_t *)user;
01192     plm_read_packets(self, self->audio_packet_type);
01193 }
01194
01195 void plm_read_packets(plm_t *self, int requested_type) {
01196     plm_packet_t *packet;
01197     while ((packet = plm_demux_decode(self->demux))) {
01198         if (packet->type == self->video_packet_type) {
01199             plm_buffer_write(self->video_buffer, packet->data, packet->length);
01200         }
01201         else if (packet->type == self->audio_packet_type) {
01202             plm_buffer_write(self->audio_buffer, packet->data, packet->length);
01203         }
01204
01205         if (packet->type == requested_type) {
01206             return;
01207         }
01208     }
01209
01210     if (plm_demux_has-ended(self->demux)) {
01211         if (self->video_buffer) {
01212             plm_buffer_signal_end(self->video_buffer);
01213         }
01214         if (self->audio_buffer) {
01215             plm_buffer_signal_end(self->audio_buffer);
01216         }
01217     }
01218 }
01219
01220 plm_frame_t *plm_seek_frame(plm_t *self, double time, int seek_exact) {
01221     if (!plm_init_decoders(self)) {
01222         return NULL;
01223     }
01224     if (!self->video_packet_type) {

```

```

01226     return NULL;
01227 }
01228
01229 int type = self->video_packet_type;
01230
01231 double start_time = plm_demux_get_start_time(self->demux, type);
01232 double duration = plm_demux_get_duration(self->demux, type);
01233
01234 if (time < 0) {
01235     time = 0;
01236 }
01237 else if (time > duration) {
01238     time = duration;
01239 }
01240
01241 plm_packet_t *packet = plm_demux_seek(self->demux, time, type, TRUE);
01242 if (!packet) {
01243     return NULL;
01244 }
01245
01246 // Disable writing to the audio buffer while decoding video
01247 int previous_audio_packet_type = self->audio_packet_type;
01248 self->audio_packet_type = 0;
01249
01250 // Clear video buffer and decode the found packet
01251 plm_video_rewind(self->video_decoder);
01252 plm_video_set_time(self->video_decoder, packet->pts - start_time);
01253 plm_buffer_write(self->video_buffer, packet->data, packet->length);
01254 plm_frame_t *frame = plm_video_decode(self->video_decoder);
01255
01256 // If we want to seek to an exact frame, we have to decode all frames
01257 // on top of the intra frame we just jumped to.
01258 if (seek_exact) {
01259     while (frame && frame->time < time) {
01260         frame = plm_video_decode(self->video_decoder);
01261     }
01262 }
01263
01264 // Enable writing to the audio buffer again?
01265 self->audio_packet_type = previous_audio_packet_type;
01266
01267 if (frame) {
01268     self->time = frame->time;
01269 }
01270
01271 self->hasEnded = FALSE;
01272 return frame;
01273 }

01274
01275 int plm_seek(plm_t *self, double time, int seek_exact) {
01276     plm_frame_t *frame = plm_seek_frame(self, time, seek_exact);
01277
01278     if (!frame) {
01279         return FALSE;
01280     }
01281
01282     if (self->video_decode_callback) {
01283         self->video_decode_callback(self, frame, self->video_decode_callback_user_data);
01284     }
01285
01286 // If audio is not enabled we are done here.
01287 if (!self->audio_packet_type) {
01288     return TRUE;
01289 }
01290
01291 // Sync up Audio. This demuxes more packets until the first audio packet
01292 // with a PTS greater than the current time is found. plm_decode() is then
01293 // called to decode enough audio data to satisfy the audio_lead_time.
01294
01295 double start_time = plm_demux_get_start_time(self->demux, self->video_packet_type);
01296 plm_audio_rewind(self->audio_decoder);
01297
01298 plm_packet_t *packet = NULL;
01299 while ((packet = plm_demux_decode(self->demux))) {
01300     if (packet->type == self->video_packet_type) {
01301         plm_buffer_write(self->video_buffer, packet->data, packet->length);
01302     }
01303     else if (
01304         packet->type == self->audio_packet_type &&
01305         packet->pts - start_time > self->time
01306     ) {
01307         plm_audio_set_time(self->audio_decoder, packet->pts - start_time);
01308         plm_buffer_write(self->audio_buffer, packet->data, packet->length);
01309         plm_decode(self, 0);
01310         break;
01311     }
01312 }

```

```

01313     return TRUE;
01314 }
01315 }
01316
01317
01318
01319 // -----
01320 // plm_buffer implementation
01321
01322 enum plm_buffer_mode {
01323     PLM_BUFFER_MODE_FILE,
01324     PLM_BUFFER_MODE_FIXED_MEM,
01325     PLM_BUFFER_MODE_RING,
01326     PLM_BUFFER_MODE_APPEND
01327 };
01328
01329 struct plm_buffer_t {
01330     size_t bit_index;
01331     size_t capacity;
01332     size_t length;
01333     size_t total_size;
01334     int discard_read_bytes;
01335     int has_ended;
01336     int free_when_done;
01337     int close_when_done;
01338     FIL *fh;
01339     plm_buffer_load_callback load_callback;
01340     void *load_callback_user_data;
01341     uint8_t *bytes;
01342     enum plm_buffer_mode mode;
01343 };
01344
01345 typedef struct {
01346     int16_t index;
01347     int16_t value;
01348 } plm_vlc_t;
01349
01350 typedef struct {
01351     int16_t index;
01352     uint16_t value;
01353 } plm_vlc_uint_t;
01354
01355
01356 void plm_buffer_seek(plm_buffer_t *self, size_t pos);
01357 size_t plm_buffer_tell(plm_buffer_t *self);
01358 void plm_buffer_discard_read_bytes(plm_buffer_t *self);
01359 void plm_buffer_load_file_callback(plm_buffer_t *self, void *user);
01360
01361 int plm_buffer_has(plm_buffer_t *self, size_t count);
01362 int plm_buffer_read(plm_buffer_t *self, int count);
01363 void plm_buffer_align(plm_buffer_t *self);
01364 void plm_buffer_skip(plm_buffer_t *self, size_t count);
01365 int plm_buffer_skip_bytes(plm_buffer_t *self, uint8_t v);
01366 int plm_buffer_next_start_code(plm_buffer_t *self);
01367 int plm_buffer_find_start_code(plm_buffer_t *self, int code);
01368 int plm_buffer_no_start_code(plm_buffer_t *self);
01369 int16_t plm_buffer_read_vlc(plm_buffer_t *self, const plm_vlc_t *table);
01370 uint16_t plm_buffer_read_vlc_uint(plm_buffer_t *self, const plm_vlc_uint_t *table);
01371
01372 plm_buffer_t *plm_buffer_create_with_filename(const char *filename) {
01373     FIL *fh = vexFileOpen(filename, "rb"); //fopen(filename, "rb");
01374     if (!fh) {
01375         return NULL;
01376     }
01377     return plm_buffer_create_with_file(fh, TRUE);
01378 }
01379
01380 plm_buffer_t *plm_buffer_create_with_file(FIL *fh, int close_when_done) {
01381     plm_buffer_t *self = plm_buffer_create_with_capacity(PLM_BUFFER_DEFAULT_SIZE);
01382     self->fh = fh;
01383     self->close_when_done = close_when_done;
01384     self->mode = PLM_BUFFER_MODE_FILE;
01385     self->discard_read_bytes = TRUE;
01386
01387     vexFileSeek(self->fh, 0, SEEK_END);
01388     self->total_size = vexFileTell(self->fh);
01389     vexFileSeek(self->fh, 0, SEEK_SET);
01390
01391     plm_buffer_set_load_callback(self, plm_buffer_load_file_callback, NULL);
01392     return self;
01393 }
01394
01395 plm_buffer_t *plm_buffer_create_with_memory(uint8_t *bytes, size_t length, int free_when_done) {
01396     plm_buffer_t *self = (plm_buffer_t *)PLM_MALLOC(sizeof(plm_buffer_t));
01397     memset(self, 0, sizeof(plm_buffer_t));
01398     self->capacity = length;
01399     self->length = length;

```

```

01400     self->total_size = length;
01401     self->free_when_done = free_when_done;
01402     self->bytes = bytes;
01403     self->mode = PLM_BUFFER_MODE_FIXED_MEM;
01404     self->discard_read_bytes = FALSE;
01405     return self;
01406 }
01407
01408 plm_buffer_t *plm_buffer_create_with_capacity(size_t capacity) {
01409     plm_buffer_t *self = (plm_buffer_t *)PLM_MALLOC(sizeof(plm_buffer_t));
01410     memset(self, 0, sizeof(plm_buffer_t));
01411     self->capacity = capacity;
01412     self->free_when_done = TRUE;
01413     self->bytes = (uint8_t *)PLM_MALLOC(capacity);
01414     self->mode = PLM_BUFFER_MODE_RING;
01415     self->discard_read_bytes = TRUE;
01416     return self;
01417 }
01418
01419 plm_buffer_t *plm_buffer_create_for_appending(size_t initial_capacity) {
01420     plm_buffer_t *self = plm_buffer_create_with_capacity(initial_capacity);
01421     self->mode = PLM_BUFFER_MODE_APEND;
01422     self->discard_read_bytes = FALSE;
01423     return self;
01424 }
01425
01426 void plm_buffer_destroy(plm_buffer_t *self) {
01427     if (self->fh && self->close_when_done) {
01428         vexFileClose(self->fh);
01429     }
01430     if (self->free_when_done) {
01431         PLM_FREE(self->bytes);
01432     }
01433     PLM_FREE(self);
01434 }
01435
01436 size_t plm_buffer_get_size(plm_buffer_t *self) {
01437     return (self->mode == PLM_BUFFER_MODE_FILE)
01438         ? self->total_size
01439         : self->length;
01440 }
01441
01442 size_t plm_buffer_get_remaining(plm_buffer_t *self) {
01443     return self->length - (self->bit_index >> 3);
01444 }
01445
01446 size_t plm_buffer_write(plm_buffer_t *self, uint8_t *bytes, size_t length) {
01447     if (self->mode == PLM_BUFFER_MODE_FIXED_MEM) {
01448         return 0;
01449     }
01450
01451     if (self->discard_read_bytes) {
01452         // This should be a ring buffer, but instead it just shifts all unread
01453         // data to the beginning of the buffer and appends new data at the end.
01454         // Seems to be good enough.
01455
01456         plm_buffer_discard_read_bytes(self);
01457         if (self->mode == PLM_BUFFER_MODE_RING) {
01458             self->total_size = 0;
01459         }
01460     }
01461
01462     // Do we have to resize to fit the new data?
01463     size_t bytes_available = self->capacity - self->length;
01464     if (bytes_available < length) {
01465         size_t new_size = self->capacity;
01466         do {
01467             new_size *= 2;
01468         } while (new_size - self->length < length);
01469         self->bytes = (uint8_t *)PLM_REALLOC(self->bytes, new_size);
01470         self->capacity = new_size;
01471     }
01472
01473     memcpy(self->bytes + self->length, bytes, length);
01474     self->length += length;
01475     self->hasEnded = FALSE;
01476     return length;
01477 }
01478
01479 void plm_buffer_signal_end(plm_buffer_t *self) {
01480     self->total_size = self->length;
01481 }
01482
01483 void plm_buffer_set_load_callback(plm_buffer_t *self, plm_buffer_load_callback fp, void *user) {
01484     self->load_callback = fp;
01485     self->load_callback_user_data = user;
01486 }
```

```

01487
01488 void plm_buffer_rewind(plm_buffer_t *self) {
01489     plm_buffer_seek(self, 0);
01490 }
01491
01492 void plm_buffer_seek(plm_buffer_t *self, size_t pos) {
01493     self->hasEnded = FALSE;
01494
01495     if (self->mode == PLM_BUFFER_MODE_FILE) {
01496         vexFileSeek(self->fh, pos, SEEK_SET);
01497         self->bit_index = 0;
01498         self->length = 0;
01499     }
01500     else if (self->mode == PLM_BUFFER_MODE_RING) {
01501         if (pos != 0) {
01502             // Seeking to non-0 is forbidden for dynamic-mem buffers
01503             return;
01504         }
01505         self->bit_index = 0;
01506         self->length = 0;
01507         self->total_size = 0;
01508     }
01509     else if (pos < self->length) {
01510         self->bit_index = pos << 3;
01511     }
01512 }
01513
01514 size_t plm_buffer_tell(plm_buffer_t *self) {
01515     return self->mode == PLM_BUFFER_MODE_FILE
01516         ? vexFileTell(self->fh) + (self->bit_index >> 3) - self->length
01517         : self->bit_index >> 3;
01518 }
01519
01520 void plm_buffer_discard_read_bytes(plm_buffer_t *self) {
01521     size_t byte_pos = self->bit_index >> 3;
01522     if (byte_pos == self->length) {
01523         self->bit_index = 0;
01524         self->length = 0;
01525     }
01526     else if (byte_pos > 0) {
01527         memmove(self->bytes, self->bytes + byte_pos, self->length - byte_pos);
01528         self->bit_index -= byte_pos << 3;
01529         self->length -= byte_pos;
01530     }
01531 }
01532
01533 void plm_buffer_load_file_callback(plm_buffer_t *self, void *user) {
01534     PLM_UNUSED(user);
01535
01536     if (self->discard_read_bytes) {
01537         plm_buffer_discard_read_bytes(self);
01538     }
01539
01540     size_t bytes_available = self->capacity - self->length;
01541     size_t bytes_read = vexFileRead((char *)self->bytes + self->length, 1, bytes_available, self->fh);
01542     self->length += bytes_read;
01543
01544     if (bytes_read == 0) {
01545         self->hasEnded = TRUE;
01546     }
01547 }
01548
01549 int plm_buffer_hasEnded(plm_buffer_t *self) {
01550     return self->hasEnded;
01551 }
01552
01553 int plm_buffer_has(plm_buffer_t *self, size_t count) {
01554     if (((self->length >> 3) - self->bit_index) >= count) {
01555         return TRUE;
01556     }
01557
01558     if (self->load_callback) {
01559         self->load_callback(self, self->load_callback_user_data);
01560
01561         if (((self->length >> 3) - self->bit_index) >= count) {
01562             return TRUE;
01563         }
01564     }
01565
01566     if (self->total_size != 0 && self->length == self->total_size) {
01567         self->hasEnded = TRUE;
01568     }
01569     return FALSE;
01570 }
01571
01572 int plm_buffer_read(plm_buffer_t *self, int count) {
01573     if (!plm_buffer_has(self, count)) {

```

```

01574     return 0;
01575 }
01576
01577 int value = 0;
01578 while (count) {
01579     int current_byte = self->bytes[self->bit_index >> 3];
01580
01581     int remaining = 8 - (self->bit_index & 7); // Remaining bits in byte
01582     int read = remaining < count ? remaining : count; // Bits in self run
01583     int shift = remaining - read;
01584     int mask = (0xff >> (8 - read));
01585
01586     value = (value << read) | ((current_byte & (mask << shift)) >> shift);
01587
01588     self->bit_index += read;
01589     count -= read;
01590 }
01591
01592 return value;
01593 }
01594
01595 void plm_buffer_align(plm_buffer_t *self) {
01596     self->bit_index = ((self->bit_index + 7) >> 3) << 3; // Align to next byte
01597 }
01598
01599 void plm_buffer_skip(plm_buffer_t *self, size_t count) {
01600     if (plm_buffer_has(self, count)) {
01601         self->bit_index += count;
01602     }
01603 }
01604
01605 int plm_buffer_skip_bytes(plm_buffer_t *self, uint8_t v) {
01606     plm_buffer_align(self);
01607     int skipped = 0;
01608     while (plm_buffer_has(self, 8) && self->bytes[self->bit_index >> 3] == v) {
01609         self->bit_index += 8;
01610         skipped++;
01611     }
01612     return skipped;
01613 }
01614
01615 int plm_buffer_next_start_code(plm_buffer_t *self) {
01616     plm_buffer_align(self);
01617
01618     while (plm_buffer_has(self, (5 << 3))) {
01619         size_t byte_index = (self->bit_index) >> 3;
01620         if (
01621             self->bytes[byte_index] == 0x00 &&
01622             self->bytes[byte_index + 1] == 0x00 &&
01623             self->bytes[byte_index + 2] == 0x01
01624         ) {
01625             self->bit_index = (byte_index + 4) << 3;
01626             return self->bytes[byte_index + 3];
01627         }
01628         self->bit_index += 8;
01629     }
01630     return -1;
01631 }
01632
01633 int plm_buffer_find_start_code(plm_buffer_t *self, int code) {
01634     int current = 0;
01635     while (TRUE) {
01636         current = plm_buffer_next_start_code(self);
01637         if (current == code || current == -1) {
01638             return current;
01639         }
01640     }
01641     return -1;
01642 }
01643
01644 int plm_buffer_has_start_code(plm_buffer_t *self, int code) {
01645     size_t previous_bit_index = self->bit_index;
01646     int previous_discard_read_bytes = self->discard_read_bytes;
01647
01648     self->discard_read_bytes = FALSE;
01649     int current = plm_buffer_find_start_code(self, code);
01650
01651     self->bit_index = previous_bit_index;
01652     self->discard_read_bytes = previous_discard_read_bytes;
01653     return current;
01654 }
01655
01656 int plm_buffer_peek_non_zero(plm_buffer_t *self, int bit_count) {
01657     if (!plm_buffer_has(self, bit_count)) {
01658         return FALSE;
01659     }
01660

```

```

01661     int val = plm_buffer_read(self, bit_count);
01662     self->bit_index -= bit_count;
01663     return val != 0;
01664 }
01665
01666 int16_t plm_buffer_read_vlc(plm_buffer_t *self, const plm_vlc_t *table) {
01667     plm_vlc_t state = {0, 0};
01668     do {
01669         state = table[state.index + plm_buffer_read(self, 1)];
01670     } while (state.index > 0);
01671     return state.value;
01672 }
01673
01674 uint16_t plm_buffer_read_vlc_uint(plm_buffer_t *self, const plm_vlc_uint_t *table) {
01675     return (uint16_t)plm_buffer_read_vlc(self, (const plm_vlc_t *)table);
01676 }
01677
01678
01679
01680 // -----
01681 // plm_demux implementation
01682
01683 static const int PLM_START_PACK = 0xBA;
01684 static const int PLM_START_END = 0xB9;
01685 static const int PLM_START_SYSTEM = 0xBB;
01686
01687 struct plm_demux_t {
01688     plm_buffer_t *buffer;
01689     int destroy_buffer_when_done;
01690     double system_clock_ref;
01691
01692     size_t last_file_size;
01693     double last_decoded_pts;
01694     double start_time;
01695     double duration;
01696
01697     int start_code;
01698     int has_pack_header;
01699     int has_system_header;
01700     int has_headers;
01701
01702     int num_audio_streams;
01703     int num_video_streams;
01704     plm_packet_t current_packet;
01705     plm_packet_t next_packet;
01706 };
01707
01708
01709 void plm_demux_buffer_seek(plm_demux_t *self, size_t pos);
01710 double plm_demux_decode_time(plm_demux_t *self);
01711 plm_packet_t *plm_demux_decode_packet(plm_demux_t *self, int type);
01712 plm_packet_t *plm_demux_get_packet(plm_demux_t *self);
01713
01714 plm_demux_t *plm_demux_create(plm_buffer_t *buffer, int destroy_when_done) {
01715     plm_demux_t *self = (plm_demux_t *)PLM_MALLOC(sizeof(plm_demux_t));
01716     memset(self, 0, sizeof(plm_demux_t));
01717
01718     self->buffer = buffer;
01719     self->destroy_buffer_when_done = destroy_when_done;
01720
01721     self->start_time = PLM_PACKET_INVALID_TS;
01722     self->duration = PLM_PACKET_INVALID_TS;
01723     self->start_code = -1;
01724
01725     plm_demux_has_headers(self);
01726     return self;
01727 }
01728
01729 void plm_demux_destroy(plm_demux_t *self) {
01730     if (self->destroy_buffer_when_done) {
01731         plm_buffer_destroy(self->buffer);
01732     }
01733     PLM_FREE(self);
01734 }
01735
01736 int plm_demux_has_headers(plm_demux_t *self) {
01737     if (self->has_headers) {
01738         return TRUE;
01739     }
01740
01741     // Decode pack header
01742     if (!self->has_pack_header) {
01743         if (
01744             self->start_code != PLM_START_PACK &&
01745             plm_buffer_find_start_code(self->buffer, PLM_START_PACK) == -1
01746         ) {
01747             return FALSE;
01748         }
01749     }
01750
01751     if (self->start_code == PLM_START_PACK) {
01752         self->has_pack_header = TRUE;
01753     }
01754
01755     if (self->start_code == PLM_START_END) {
01756         self->has_system_header = TRUE;
01757     }
01758
01759     if (self->start_code == PLM_START_SYSTEM) {
01760         self->has_headers = TRUE;
01761     }
01762
01763     if (self->has_headers) {
01764         self->has_system_header = TRUE;
01765     }
01766
01767     if (self->has_system_header) {
01768         self->has_pack_header = TRUE;
01769     }
01770
01771     if (self->has_pack_header) {
01772         self->has_headers = TRUE;
01773     }
01774
01775     if (self->has_headers) {
01776         self->has_system_header = TRUE;
01777     }
01778
01779     if (self->has_system_header) {
01780         self->has_pack_header = TRUE;
01781     }
01782
01783     if (self->has_pack_header) {
01784         self->has_headers = TRUE;
01785     }
01786
01787     if (self->has_headers) {
01788         self->has_system_header = TRUE;
01789     }
01790
01791     if (self->has_system_header) {
01792         self->has_pack_header = TRUE;
01793     }
01794
01795     if (self->has_pack_header) {
01796         self->has_headers = TRUE;
01797     }
01798
01799     if (self->has_headers) {
01800         self->has_system_header = TRUE;
01801     }
01802
01803     if (self->has_system_header) {
01804         self->has_pack_header = TRUE;
01805     }
01806
01807     if (self->has_pack_header) {
01808         self->has_headers = TRUE;
01809     }
01810
01811     if (self->has_headers) {
01812         self->has_system_header = TRUE;
01813     }
01814
01815     if (self->has_system_header) {
01816         self->has_pack_header = TRUE;
01817     }
01818
01819     if (self->has_pack_header) {
01820         self->has_headers = TRUE;
01821     }
01822
01823     if (self->has_headers) {
01824         self->has_system_header = TRUE;
01825     }
01826
01827     if (self->has_system_header) {
01828         self->has_pack_header = TRUE;
01829     }
01830
01831     if (self->has_pack_header) {
01832         self->has_headers = TRUE;
01833     }
01834
01835     if (self->has_headers) {
01836         self->has_system_header = TRUE;
01837     }
01838
01839     if (self->has_system_header) {
01840         self->has_pack_header = TRUE;
01841     }
01842
01843     if (self->has_pack_header) {
01844         self->has_headers = TRUE;
01845     }
01846
01847     if (self->has_headers) {
01848         self->has_system_header = TRUE;
01849     }
01850
01851     if (self->has_system_header) {
01852         self->has_pack_header = TRUE;
01853     }
01854
01855     if (self->has_pack_header) {
01856         self->has_headers = TRUE;
01857     }
01858
01859     if (self->has_headers) {
01860         self->has_system_header = TRUE;
01861     }
01862
01863     if (self->has_system_header) {
01864         self->has_pack_header = TRUE;
01865     }
01866
01867     if (self->has_pack_header) {
01868         self->has_headers = TRUE;
01869     }
01870
01871     if (self->has_headers) {
01872         self->has_system_header = TRUE;
01873     }
01874
01875     if (self->has_system_header) {
01876         self->has_pack_header = TRUE;
01877     }
01878
01879     if (self->has_pack_header) {
01880         self->has_headers = TRUE;
01881     }
01882
01883     if (self->has_headers) {
01884         self->has_system_header = TRUE;
01885     }
01886
01887     if (self->has_system_header) {
01888         self->has_pack_header = TRUE;
01889     }
01890
01891     if (self->has_pack_header) {
01892         self->has_headers = TRUE;
01893     }
01894
01895     if (self->has_headers) {
01896         self->has_system_header = TRUE;
01897     }
01898
01899     if (self->has_system_header) {
01900         self->has_pack_header = TRUE;
01901     }
01902
01903     if (self->has_pack_header) {
01904         self->has_headers = TRUE;
01905     }
01906
01907     if (self->has_headers) {
01908         self->has_system_header = TRUE;
01909     }
01910
01911     if (self->has_system_header) {
01912         self->has_pack_header = TRUE;
01913     }
01914
01915     if (self->has_pack_header) {
01916         self->has_headers = TRUE;
01917     }
01918
01919     if (self->has_headers) {
01920         self->has_system_header = TRUE;
01921     }
01922
01923     if (self->has_system_header) {
01924         self->has_pack_header = TRUE;
01925     }
01926
01927     if (self->has_pack_header) {
01928         self->has_headers = TRUE;
01929     }
01930
01931     if (self->has_headers) {
01932         self->has_system_header = TRUE;
01933     }
01934
01935     if (self->has_system_header) {
01936         self->has_pack_header = TRUE;
01937     }
01938
01939     if (self->has_pack_header) {
01940         self->has_headers = TRUE;
01941     }
01942
01943     if (self->has_headers) {
01944         self->has_system_header = TRUE;
01945     }
01946
01947     if (self->has_system_header) {
01948         self->has_pack_header = TRUE;
01949     }
01950
01951     if (self->has_pack_header) {
01952         self->has_headers = TRUE;
01953     }
01954
01955     if (self->has_headers) {
01956         self->has_system_header = TRUE;
01957     }
01958
01959     if (self->has_system_header) {
01960         self->has_pack_header = TRUE;
01961     }
01962
01963     if (self->has_pack_header) {
01964         self->has_headers = TRUE;
01965     }
01966
01967     if (self->has_headers) {
01968         self->has_system_header = TRUE;
01969     }
01970
01971     if (self->has_system_header) {
01972         self->has_pack_header = TRUE;
01973     }
01974
01975     if (self->has_pack_header) {
01976         self->has_headers = TRUE;
01977     }
01978
01979     if (self->has_headers) {
01980         self->has_system_header = TRUE;
01981     }
01982
01983     if (self->has_system_header) {
01984         self->has_pack_header = TRUE;
01985     }
01986
01987     if (self->has_pack_header) {
01988         self->has_headers = TRUE;
01989     }
01990
01991     if (self->has_headers) {
01992         self->has_system_header = TRUE;
01993     }
01994
01995     if (self->has_system_header) {
01996         self->has_pack_header = TRUE;
01997     }
01998
01999     if (self->has_pack_header) {
02000         self->has_headers = TRUE;
02001     }
02002
02003     if (self->has_headers) {
02004         self->has_system_header = TRUE;
02005     }
02006
02007     if (self->has_system_header) {
02008         self->has_pack_header = TRUE;
02009     }
02010
02011     if (self->has_pack_header) {
02012         self->has_headers = TRUE;
02013     }
02014
02015     if (self->has_headers) {
02016         self->has_system_header = TRUE;
02017     }
02018
02019     if (self->has_system_header) {
02020         self->has_pack_header = TRUE;
02021     }
02022
02023     if (self->has_pack_header) {
02024         self->has_headers = TRUE;
02025     }
02026
02027     if (self->has_headers) {
02028         self->has_system_header = TRUE;
02029     }
02030
02031     if (self->has_system_header) {
02032         self->has_pack_header = TRUE;
02033     }
02034
02035     if (self->has_pack_header) {
02036         self->has_headers = TRUE;
02037     }
02038
02039     if (self->has_headers) {
02040         self->has_system_header = TRUE;
02041     }
02042
02043     if (self->has_system_header) {
02044         self->has_pack_header = TRUE;
02045     }
02046
02047     if (self->has_pack_header) {
02048         self->has_headers = TRUE;
02049     }
02050
02051     if (self->has_headers) {
02052         self->has_system_header = TRUE;
02053     }
02054
02055     if (self->has_system_header) {
02056         self->has_pack_header = TRUE;
02057     }
02058
02059     if (self->has_pack_header) {
02060         self->has_headers = TRUE;
02061     }
02062
02063     if (self->has_headers) {
02064         self->has_system_header = TRUE;
02065     }
02066
02067     if (self->has_system_header) {
02068         self->has_pack_header = TRUE;
02069     }
02070
02071     if (self->has_pack_header) {
02072         self->has_headers = TRUE;
02073     }
02074
02075     if (self->has_headers) {
02076         self->has_system_header = TRUE;
02077     }
02078
02079     if (self->has_system_header) {
02080         self->has_pack_header = TRUE;
02081     }
02082
02083     if (self->has_pack_header) {
02084         self->has_headers = TRUE;
02085     }
02086
02087     if (self->has_headers) {
02088         self->has_system_header = TRUE;
02089     }
02090
02091     if (self->has_system_header) {
02092         self->has_pack_header = TRUE;
02093     }
02094
02095     if (self->has_pack_header) {
02096         self->has_headers = TRUE;
02097     }
02098
02099     if (self->has_headers) {
02100         self->has_system_header = TRUE;
02101     }
02102
02103     if (self->has_system_header) {
02104         self->has_pack_header = TRUE;
02105     }
02106
02107     if (self->has_pack_header) {
02108         self->has_headers = TRUE;
02109     }
02110
02111     if (self->has_headers) {
02112         self->has_system_header = TRUE;
02113     }
02114
02115     if (self->has_system_header) {
02116         self->has_pack_header = TRUE;
02117     }
02118
02119     if (self->has_pack_header) {
02120         self->has_headers = TRUE;
02121     }
02122
02123     if (self->has_headers) {
02124         self->has_system_header = TRUE;
02125     }
02126
02127     if (self->has_system_header) {
02128         self->has_pack_header = TRUE;
02129     }
02130
02131     if (self->has_pack_header) {
02132         self->has_headers = TRUE;
02133     }
02134
02135     if (self->has_headers) {
02136         self->has_system_header = TRUE;
02137     }
02138
02139     if (self->has_system_header) {
02140         self->has_pack_header = TRUE;
02141     }
02142
02143     if (self->has_pack_header) {
02144         self->has_headers = TRUE;
02145     }
02146
02147     if (self->has_headers) {
02148         self->has_system_header = TRUE;
02149     }
02150
02151     if (self->has_system_header) {
02152         self->has_pack_header = TRUE;
02153     }
02154
02155     if (self->has_pack_header) {
02156         self->has_headers = TRUE;
02157     }
02158
02159     if (self->has_headers) {
02160         self->has_system_header = TRUE;
02161     }
02162
02163     if (self->has_system_header) {
02164         self->has_pack_header = TRUE;
02165     }
02166
02167     if (self->has_pack_header) {
02168         self->has_headers = TRUE;
02169     }
02170
02171     if (self->has_headers) {
02172         self->has_system_header = TRUE;
02173     }
02174
02175     if (self->has_system_header) {
02176         self->has_pack_header = TRUE;
02177     }
02178
02179     if (self->has_pack_header) {
02180         self->has_headers = TRUE;
02181     }
02182
02183     if (self->has_headers) {
02184         self->has_system_header = TRUE;
02185     }
02186
02187     if (self->has_system_header) {
02188         self->has_pack_header = TRUE;
02189     }
02190
02191     if (self->has_pack_header) {
02192         self->has_headers = TRUE;
02193     }
02194
02195     if (self->has_headers) {
02196         self->has_system_header = TRUE;
02197     }
02198
02199     if (self->has_system_header) {
02200         self->has_pack_header = TRUE;
02201     }
02202
02203     if (self->has_pack_header) {
02204         self->has_headers = TRUE;
02205     }
02206
02207     if (self->has_headers) {
02208         self->has_system_header = TRUE;
02209     }
02210
02211     if (self->has_system_header) {
02212         self->has_pack_header = TRUE;
02213     }
02214
02215     if (self->has_pack_header) {
02216         self->has_headers = TRUE;
02217     }
02218
02219     if (self->has_headers) {
02220         self->has_system_header = TRUE;
02221     }
02222
02223     if (self->has_system_header) {
02224         self->has_pack_header = TRUE;
02225     }
02226
02227     if (self->has_pack_header) {
02228         self->has_headers = TRUE;
02229     }
02230
02231     if (self->has_headers) {
02232         self->has_system_header = TRUE;
02233     }
02234
02235     if (self->has_system_header) {
02236         self->has_pack_header = TRUE;
02237     }
02238
02239     if (self->has_pack_header) {
02240         self->has_headers = TRUE;
02241     }
02242
02243     if (self->has_headers) {
02244         self->has_system_header = TRUE;
02245     }
02246
02247     if (self->has_system_header) {
02248         self->has_pack_header = TRUE;
02249     }
02250
02251     if (self->has_pack_header) {
02252         self->has_headers = TRUE;
02253     }
02254
02255     if (self->has_headers) {
02256         self->has_system_header = TRUE;
02257     }
02258
02259     if (self->has_system_header) {
02260         self->has_pack_header = TRUE;
02261     }
02262
02263     if (self->has_pack_header) {
02264         self->has_headers = TRUE;
02265     }
02266
02267     if (self->has_headers) {
02268         self->has_system_header = TRUE;
02269     }
02270
02271     if (self->has_system_header) {
02272         self->has_pack_header = TRUE;
02273     }
02274
02275     if (self->has_pack_header) {
02276         self->has_headers = TRUE;
02277     }
02278
02279     if (self->has_headers) {
02280         self->has_system_header = TRUE;
02281     }
02282
02283     if (self->has_system_header) {
02284         self->has_pack_header = TRUE;
02285     }
02286
02287     if (self->has_pack_header) {
02288         self->has_headers = TRUE;
02289     }
02290
02291     if (self->has_headers) {
02292         self->has_system_header = TRUE;
02293     }
02294
02295     if (self->has_system_header) {
02296         self->has_pack_header = TRUE;
02297     }
02298
02299     if (self->has_pack_header) {
02300         self->has_headers = TRUE;
02301     }
02302
02303     if (self->has_headers) {
02304         self->has_system_header = TRUE;
02305     }
02306
02307     if (self->has_system_header) {
02308         self->has_pack_header = TRUE;
02309     }
02310
02311     if (self->has_pack_header) {
02312         self->has_headers = TRUE;
02313     }
02314
02315     if (self->has_headers) {
02316         self->has_system_header = TRUE;
02317     }
02318
02319     if (self->has_system_header) {
02320         self->has_pack_header = TRUE;
02321     }
02322
02323     if (self->has_pack_header) {
02324         self->has_headers = TRUE;
02325     }
02326
02327     if (self->has_headers) {
02328         self->has_system_header = TRUE;
02329     }
02330
02331     if (self->has_system_header) {
02332         self->has_pack_header = TRUE;
02333     }
02334
02335     if (self->has_pack_header) {
02336         self->has_headers = TRUE;
02337     }
02338
02339     if (self->has_headers) {
02340         self->has_system_header = TRUE;
02341     }
02342
02343     if (self->has_system_header) {
02344         self->has_pack_header = TRUE;
02345     }
02346
02347     if (self->has_pack_header) {
02348         self->has_headers = TRUE;
02349     }
02350
02351     if (self->has_headers) {
02352         self->has_system_header = TRUE;
02353     }
02354
02355     if (self->has_system_header) {
02356         self->has_pack_header = TRUE;
02357     }
02358
02359     if (self->has_pack_header) {
02360         self->has_headers = TRUE;
02361     }
02362
02363     if (self->has_headers) {
02364         self->has_system_header = TRUE;
02365     }
02366
02367     if (self->has_system_header) {
02368         self->has_pack_header = TRUE;
02369     }
02370
02371     if (self->has_pack_header) {
02372         self->has_headers = TRUE;
02373     }
02374
02375     if (self->has_headers) {
02376         self->has_system_header = TRUE;
02377     }
02378
02379     if (self->has_system_header) {
02380         self->has_pack_header = TRUE;
02381     }
02382
02383     if (self->has_pack_header) {
02384         self->has_headers = TRUE;
02385     }
02386
02387     if (self->has_headers) {
02388         self->has_system_header = TRUE;
02389     }
02390
02391     if (self->has_system_header) {
02392         self->has_pack_header = TRUE;
02393     }
02394
02395     if (self->has_pack_header) {
02396         self->has_headers = TRUE;
02397     }
02398
02399     if (self->has_headers) {
02400         self->has_system_header = TRUE;
02401     }
02402
02403     if (self->has_system_header) {
02404         self->has_pack_header = TRUE;
02405     }
02406
02407     if (self->has_pack_header) {
02408         self->has_headers = TRUE;
02409     }
02410
02411     if (self->has_headers) {
02412         self->has_system_header = TRUE;
02413     }
02414
02415     if (self->has_system_header) {
02416         self->has_pack_header = TRUE;
02417     }
02418
02419     if (self->has_pack_header) {
02420         self->has_headers = TRUE;
02421     }
02422
02423     if (self->has_headers) {
02424         self->has_system_header = TRUE;
02425     }
02426
02427     if (self->has_system_header) {
02428         self->has_pack_header = TRUE;
02429     }
02430
02431     if (self->has_pack_header) {
02432         self->has_headers = TRUE;
02433     }
02434
02435     if (self->has_headers) {
02436         self->has_system_header = TRUE;
02437     }
02438
02439     if (self->has_system_header) {
02440         self->has_pack_header = TRUE;
02441     }
02442
02443     if (self->has_pack_header) {
02444         self->has_headers = TRUE;
02445     }
02446
02447     if (self->has_headers) {
02448         self->has_system_header = TRUE;
02449     }
02450
02451     if (self->has_system_header) {
02452         self->has_pack_header = TRUE;
02453     }
02454
02455     if (self->has_pack_header) {
02456         self->has_headers = TRUE;
02457     }
02458
02459     if (self->has_headers) {
02460         self->has_system_header = TRUE;
02461     }
02462
02463     if (self->has_system_header) {
02464         self->has_pack_header = TRUE;
02465     }
02466
02467     if (self->has_pack_header) {
02468         self->has_headers = TRUE;
02469     }
02470
02471     if (self->has_headers) {
02472         self->has_system_header = TRUE;
02473     }
02474
02475     if (self->has_system_header) {
02476         self->has_pack_header = TRUE;
02477     }
02478
02479     if (self->has_pack_header) {
02480         self->has_headers = TRUE;
02481     }
02482
02483     if (self->has_headers) {
02484         self->has_system_header = TRUE;
02485     }
02486
02487     if (self->has_system_header) {
02488         self->has_pack_header = TRUE;
02489     }
02490
02491     if (self->has_pack_header) {
02492         self->has_headers = TRUE;
02493     }
02494
02495     if (self->has_headers) {
02496         self->has_system_header = TRUE;
02497     }
02498
02499     if (self->has_system_header) {
02500         self->has_pack_header = TRUE;
02501     }
02502
02503     if (self->has_pack_header) {
02504         self->has_headers = TRUE;
02505     }
02506
02507     if (self->has_headers) {
02508         self->has_system_header = TRUE;
02509     }
02510
02511     if (self->has_system_header) {
02512         self->has_pack_header = TRUE;
02513     }
02514
02515     if (self->has_pack_header) {
02516         self->has_headers = TRUE;
02517     }
02518
02519     if (self->has_headers) {
02520         self->has_system_header = TRUE;
02521     }
02522
02523     if (self->has_system_header) {
02524         self->has_pack_header = TRUE;
02525     }
02526
02527     if (self->has_pack_header) {
02528         self->has_headers = TRUE;
02529     }
02530
02531     if (self->has_headers) {
02532         self->has_system_header = TRUE;
02533     }
02534
02535     if (self->has_system_header) {
02536         self->has_pack_header = TRUE;
02537     }
02538
02539     if (self->has_pack_header) {
02540         self->has_headers = TRUE;
02541     }
02542
02543     if (self->has_headers) {
02544         self->has_system_header = TRUE;
02545     }
02546
02547     if (self->has_system_header) {
02548         self->has_pack_header = TRUE;
02549     }
02550
02551     if (self->has_pack_header) {
02552         self->has_headers = TRUE;
02553     }
02554
02555     if (self->has_headers) {
02556         self->has_system_header = TRUE;
02557     }
02558
02559     if (self->has_system_header) {
02560         self->has_pack_header = TRUE;
02561     }
02562
02563     if (self->has_pack_header) {
02564         self->has_headers = TRUE;
02565     }
02566
02567     if (self->has_headers) {
02568         self->has_system_header = TRUE;
02569     }
02570
02571     if (self->has_system_header) {
02572         self->has_pack_header = TRUE;
02573     }
02574
02575     if (self->has_pack_header) {
02576         self->has_headers = TRUE;
02577     }
02578
02579     if (self->has_headers) {
02580         self->has_system_header = TRUE;
02581     }
02582
02583     if (self->has_system_header) {
02584         self->has_pack_header = TRUE;
02585     }
02586
02587     if (self->has_pack_header) {
02588         self->has_headers = TRUE;
02589     }
02590
02591     if (self->has_headers) {
02592         self->has_system_header = TRUE;
02593     }
02594
02595     if (self->has_system_header) {
02596         self->has_pack_header = TRUE;
02597     }
02598
02599     if (self->has_pack_header) {
02600         self->has_headers = TRUE;
02601     }
02602
02603     if (self->has_headers) {
02604         self->has_system_header = TRUE;
02605     }
02606
02607     if (self->has_system_header) {
02608         self->has_pack_header = TRUE;
02609     }
02610
02611     if (self->has_pack_header) {
02612         self->has_headers = TRUE;
02613     }
02614
02615     if (self->has_headers) {
02616         self->has_system_header = TRUE;
02617     }
02618
02619     if (self->has_system_header) {
02620         self->has_pack_header = TRUE;
02621     }
02622
02623     if (self->has_pack_header) {
02624         self->has_headers = TRUE;
02625     }
02626
0
```

```

01748         }
01749
01750     self->start_code = PLM_START_PACK;
01751     if (!plm_buffer_has(self->buffer, 64)) {
01752         return FALSE;
01753     }
01754     self->start_code = -1;
01755
01756     if (plm_buffer_read(self->buffer, 4) != 0x02) {
01757         return FALSE;
01758     }
01759
01760     self->system_clock_ref = plm_demux_decode_time(self);
01761     plm_buffer_skip(self->buffer, 1);
01762     plm_buffer_skip(self->buffer, 22); // mux_rate * 50
01763     plm_buffer_skip(self->buffer, 1);
01764
01765     self->has_pack_header = TRUE;
01766 }
01767
01768 // Decode system header
01769 if (!self->has_system_header) {
01770     if (
01771         self->start_code != PLM_START_SYSTEM &&
01772         plm_buffer_find_start_code(self->buffer, PLM_START_SYSTEM) == -1
01773     ) {
01774         return FALSE;
01775     }
01776
01777     self->start_code = PLM_START_SYSTEM;
01778     if (!plm_buffer_has(self->buffer, 56)) {
01779         return FALSE;
01780     }
01781     self->start_code = -1;
01782
01783     plm_buffer_skip(self->buffer, 16); // header_length
01784     plm_buffer_skip(self->buffer, 24); // rate bound
01785     self->num_audio_streams = plm_buffer_read(self->buffer, 6);
01786     plm_buffer_skip(self->buffer, 5); // misc flags
01787     self->num_video_streams = plm_buffer_read(self->buffer, 5);
01788
01789     self->has_system_header = TRUE;
01790 }
01791
01792 self->has_headers = TRUE;
01793 return TRUE;
01794 }
01795
01796 int plm_demux_get_num_video_streams(plm_demux_t *self) {
01797     return plm_demux_has_headers(self)
01798         ? self->num_video_streams
01799         : 0;
01800 }
01801
01802 int plm_demux_get_num_audio_streams(plm_demux_t *self) {
01803     return plm_demux_has_headers(self)
01804         ? self->num_audio_streams
01805         : 0;
01806 }
01807
01808 void plm_demux_rewind(plm_demux_t *self) {
01809     plm_buffer_rewind(self->buffer);
01810     self->current_packet.length = 0;
01811     self->next_packet.length = 0;
01812     self->start_code = -1;
01813 }
01814
01815 int plm_demux_has-ended(plm_demux_t *self) {
01816     return plm_buffer_has-ended(self->buffer);
01817 }
01818
01819 void plm_demux_buffer_seek(plm_demux_t *self, size_t pos) {
01820     plm_buffer_seek(self->buffer, pos);
01821     self->current_packet.length = 0;
01822     self->next_packet.length = 0;
01823     self->start_code = -1;
01824 }
01825
01826 double plm_demux_get_start_time(plm_demux_t *self, int type) {
01827     if (self->start_time != PLM_PACKET_INVALID_TS) {
01828         return self->start_time;
01829     }
01830
01831     int previous_pos = plm_buffer_tell(self->buffer);
01832     int previous_start_code = self->start_code;
01833
01834     // Find first video PTS

```

```

01835     plm_demux_rewind(self);
01836     do {
01837         plm_packet_t *packet = plm_demux_decode(self);
01838         if (!packet) {
01839             break;
01840         }
01841         if (packet->type == type) {
01842             self->start_time = packet->pts;
01843         }
01844     } while (self->start_time == PLM_PACKET_INVALID_TS);
01845
01846     plm_demux_buffer_seek(self, previous_pos);
01847     self->start_code = previous_start_code;
01848     return self->start_time;
01849 }
01850
01851 double plm_demux_get_duration(plm_demux_t *self, int type) {
01852     size_t file_size = plm_buffer_get_size(self->buffer);
01853
01854     if (
01855         self->duration != PLM_PACKET_INVALID_TS &&
01856         self->last_file_size == file_size
01857     ) {
01858         return self->duration;
01859     }
01860
01861     size_t previous_pos = plm_buffer_tell(self->buffer);
01862     int previous_start_code = self->start_code;
01863
01864     // Find last video PTS. Start searching 64kb from the end and go further
01865     // back if needed.
01866     long start_range = 64 * 1024;
01867     long max_range = 4096 * 1024;
01868     for (long range = start_range; range <= max_range; range *= 2) {
01869         long seek_pos = file_size - range;
01870         if (seek_pos < 0) {
01871             seek_pos = 0;
01872             range = max_range; // Make sure to bail after this round
01873         }
01874         plm_demux_buffer_seek(self, seek_pos);
01875         self->current_packet.length = 0;
01876
01877         double last_pts = PLM_PACKET_INVALID_TS;
01878         plm_packet_t *packet = NULL;
01879         while ((packet = plm_demux_decode(self))) {
01880             if (packet->pts != PLM_PACKET_INVALID_TS && packet->type == type) {
01881                 last_pts = packet->pts;
01882             }
01883         }
01884         if (last_pts != PLM_PACKET_INVALID_TS) {
01885             self->duration = last_pts - plm_demux_get_start_time(self, type);
01886             break;
01887         }
01888     }
01889
01890     plm_demux_buffer_seek(self, previous_pos);
01891     self->start_code = previous_start_code;
01892     self->last_file_size = file_size;
01893     return self->duration;
01894 }
01895
01896 plm_packet_t *plm_demux_seek(plm_demux_t *self, double seek_time, int type, int force_intra) {
01897     if (!plm_demux_has_headers(self)) {
01898         return NULL;
01899     }
01900
01901     // Using the current time, current byte position and the average bytes per
01902     // second for this file, try to jump to a byte position that hopefully has
01903     // packets containing timestamps within one second before to the desired
01904     // seek_time.
01905
01906     // If we hit close to the seek_time scan through all packets to find the
01907     // last one (just before the seek_time) containing an intra frame.
01908     // Otherwise we should at least be closer than before. Calculate the bytes
01909     // per second for the jumped range and jump again.
01910
01911     // The number of retries here is hard-limited to a generous amount. Usually
01912     // the correct range is found after 1-5 jumps, even for files with very
01913     // variable bitrates. If significantly more jumps are needed, there's
01914     // probably something wrong with the file and we just avoid getting into an
01915     // infinite loop. 32 retries should be enough for anybody.
01916
01917     double duration = plm_demux_get_duration(self, type);
01918     long file_size = plm_buffer_get_size(self->buffer);
01919     long byterate = file_size / duration;
01920
01921     double cur_time = self->last_decoded_pts;

```

```

01922     double scan_span = 1;
01923
01924     if (seek_time > duration) {
01925         seek_time = duration;
01926     }
01927     else if (seek_time < 0) {
01928         seek_time = 0;
01929     }
01930     seek_time += self->start_time;
01931
01932     for (int retry = 0; retry < 32; retry++) {
01933         int found_packet_with_pts = FALSE;
01934         int found_packet_in_range = FALSE;
01935         long last_valid_packet_start = -1;
01936         double first_packet_time = PLM_PACKET_INVALID_TS;
01937
01938         long cur_pos = plm_buffer_tell(self->buffer);
01939
01940         // Estimate byte offset and jump to it.
01941         long offset = (seek_time - cur_time - scan_span) * byterate;
01942         long seek_pos = cur_pos + offset;
01943         if (seek_pos < 0) {
01944             seek_pos = 0;
01945         }
01946         else if (seek_pos > file_size - 256) {
01947             seek_pos = file_size - 256;
01948         }
01949
01950         plm_demux_buffer_seek(self, seek_pos);
01951
01952         // Scan through all packets up to the seek_time to find the last packet
01953         // containing an intra frame.
01954         while (plm_buffer_find_start_code(self->buffer, type) != -1) {
01955             long packet_start = plm_buffer_tell(self->buffer);
01956             plm_packet_t *packet = plm_demux_decode_packet(self, type);
01957
01958             // Skip packet if it has no PTS
01959             if (!packet || packet->pts == PLM_PACKET_INVALID_TS) {
01960                 continue;
01961             }
01962
01963             // Bail scanning through packets if we hit one that is outside
01964             // seek_time - scan_span.
01965             // We also adjust the cur_time and byterate values here so the next
01966             // iteration can be a bit more precise.
01967             if (packet->pts > seek_time || packet->pts < seek_time - scan_span) {
01968                 found_packet_with_pts = TRUE;
01969                 byterate = (seek_pos - cur_pos) / (packet->pts - cur_time);
01970                 cur_time = packet->pts;
01971                 break;
01972             }
01973
01974             // If we are still here, it means this packet is in close range to
01975             // the seek_time. If this is the first packet for this jump position
01976             // record the PTS. If we later have to back off, when there was no
01977             // intra frame in this range, we can lower the seek_time to not scan
01978             // this range again.
01979             if (!found_packet_in_range) {
01980                 found_packet_in_range = TRUE;
01981                 first_packet_time = packet->pts;
01982             }
01983
01984             // Check if this is an intra frame packet. If so, record the buffer
01985             // position of the start of this packet. We want to jump back to it
01986             // later, when we know it's the last intra frame before desired
01987             // seek time.
01988             if (force_intra) {
01989                 for (size_t i = 0; i < packet->length - 6; i++) {
01990                     // Find the START_PICTURE code
01991                     if (
01992                         packet->data[i] == 0x00 &&
01993                         packet->data[i + 1] == 0x00 &&
01994                         packet->data[i + 2] == 0x01 &&
01995                         packet->data[i + 3] == 0x00
01996                     ) {
01997                         // Bits 11--13 in the picture header contain the frame
01998                         // type, where 1=Intra
01999                         if ((packet->data[i + 5] & 0x38) == 8) {
02000                             last_valid_packet_start = packet_start;
02001                         }
02002                         break;
02003                     }
02004                 }
02005             }
02006
02007             // If we don't want intra frames, just use the last PTS found.
02008             else {

```

```

02009         last_valid_packet_start = packet_start;
02010     }
02011 }
02012
02013 // If there was at least one intra frame in the range scanned above,
02014 // our search is over. Jump back to the packet and decode it again.
02015 if (last_valid_packet_start != -1) {
02016     plm_demux_buffer_seek(self, last_valid_packet_start);
02017     return plm_demux_decode_packet(self, type);
02018 }
02019
02020 // If we hit the right range, but still found no intra frame, we have
02021 // to increase the scan_span. This is done exponentially to also handle
02022 // video files with very few intra frames.
02023 else if (found_packet_in_range) {
02024     scan_span *= 2;
02025     seek_time = first_packet_time;
02026 }
02027
02028 // If we didn't find any packet with a PTS, it probably means we reached
02029 // the end of the file. Estimate byterate and cur_time accordingly.
02030 else if (!found_packet_with_pts) {
02031     byterate = (seek_pos - cur_pos) / (duration - cur_time);
02032     cur_time = duration;
02033 }
02034 }
02035
02036 return NULL;
02037 }
02038
02039 plm_packet_t *plm_demux_decode(plm_demux_t *self) {
02040     if (!plm_demux_has_headers(self)) {
02041         return NULL;
02042     }
02043
02044     if (self->current_packet.length) {
02045         size_t bits_till_next_packet = self->current_packet.length << 3;
02046         if (!plm_buffer_has(self->buffer, bits_till_next_packet)) {
02047             return NULL;
02048         }
02049         plm_buffer_skip(self->buffer, bits_till_next_packet);
02050         self->current_packet.length = 0;
02051     }
02052
02053 // Pending packet waiting for data?
02054 if (self->next_packet.length) {
02055     return plm_demux_get_packet(self);
02056 }
02057
02058 // Pending packet waiting for header?
02059 if (self->start_code != -1) {
02060     return plm_demux_decode_packet(self, self->start_code);
02061 }
02062
02063 do {
02064     self->start_code = plm_buffer_next_start_code(self->buffer);
02065     if (
02066         self->start_code == PLM_DEMUX_PACKET_VIDEO_1 ||
02067         self->start_code == PLM_DEMUX_PACKET_PRIVATE ||
02068         self->start_code >= PLM_DEMUX_PACKET_AUDIO_1 &&
02069         self->start_code <= PLM_DEMUX_PACKET_AUDIO_4
02070     )
02071     ) {
02072         return plm_demux_decode_packet(self, self->start_code);
02073     }
02074 } while (self->start_code != -1);
02075
02076 return NULL;
02077 }
02078
02079 double plm_demux_decode_time(plm_demux_t *self) {
02080     int64_t clock = plm_buffer_read(self->buffer, 3) << 30;
02081     plm_buffer_skip(self->buffer, 1);
02082     clock |= plm_buffer_read(self->buffer, 15) << 15;
02083     plm_buffer_skip(self->buffer, 1);
02084     clock |= plm_buffer_read(self->buffer, 15);
02085     plm_buffer_skip(self->buffer, 1);
02086     return (double)clock / 90000.0;
02087 }
02088
02089 plm_packet_t *plm_demux_decode_packet(plm_demux_t *self, int type) {
02090     if (!plm_buffer_has(self->buffer, 16 << 3)) {
02091         return NULL;
02092     }
02093
02094     self->start_code = -1;
02095

```

```

02096     self->next_packet.type = type;
02097     self->next_packet.length = plm_buffer_read(self->buffer, 16);
02098     self->next_packet.length -= plm_buffer_skip_bytes(self->buffer, 0xff); // stuffing
02099
02100     // skip P-STD
02101     if (plm_buffer_read(self->buffer, 2) == 0x01) {
02102         plm_buffer_skip(self->buffer, 16);
02103         self->next_packet.length -= 2;
02104     }
02105
02106     int pts_dts_marker = plm_buffer_read(self->buffer, 2);
02107     if (pts_dts_marker == 0x03) {
02108         self->next_packet.pts = plm_demux_decode_time(self);
02109         self->last_decoded_pts = self->next_packet.pts;
02110         plm_buffer_skip(self->buffer, 40); // skip dts
02111         self->next_packet.length -= 10;
02112     }
02113     else if (pts_dts_marker == 0x02) {
02114         self->next_packet.pts = plm_demux_decode_time(self);
02115         self->last_decoded_pts = self->next_packet.pts;
02116         self->next_packet.length -= 5;
02117     }
02118     else if (pts_dts_marker == 0x00) {
02119         self->next_packet.pts = PLM_PACKET_INVALID_TS;
02120         plm_buffer_skip(self->buffer, 4);
02121         self->next_packet.length -= 1;
02122     }
02123     else {
02124         return NULL; // invalid
02125     }
02126
02127     return plm_demux_get_packet(self);
02128 }
02129
02130 plm_packet_t *plm_demux_get_packet(plm_demux_t *self) {
02131     if (!plm_buffer_has(self->buffer, self->next_packet.length << 3)) {
02132         return NULL;
02133     }
02134
02135     self->current_packet.data = self->buffer->bytes + (self->buffer->bit_index >> 3);
02136     self->current_packet.length = self->next_packet.length;
02137     self->current_packet.type = self->next_packet.type;
02138     self->current_packet.pts = self->next_packet.pts;
02139
02140     self->next_packet.length = 0;
02141     return &self->current_packet;
02142 }
02143
02144
02145
02146 // -----
02147 // plm_video implementation
02148
02149 // Inspired by Java MPEG-1 Video Decoder and Player by Zoltan Korandi
02150 // https://sourceforge.net/projects/javampeg1video/
02151
02152 static const int PLM_VIDEO_PICTURE_TYPE_INTRA = 1;
02153 static const int PLM_VIDEO_PICTURE_TYPE_PREDICTIVE = 2;
02154 static const int PLM_VIDEO_PICTURE_TYPE_B = 3;
02155
02156 static const int PLM_START_SEQUENCE = 0xB3;
02157 static const int PLM_START_SLICE_FIRST = 0x01;
02158 static const int PLM_START_SLICE_LAST = 0xAF;
02159 static const int PLM_START_PICTURE = 0x00;
02160 static const int PLM_START_EXTENSION = 0xB5;
02161 static const int PLM_START_USER_DATA = 0xB2;
02162
02163 #define PLM_START_IS_SLICE(c) \
02164     (c >= PLM_START_SLICE_FIRST && c <= PLM_START_SLICE_LAST)
02165
02166 static const double PLM_VIDEO_PICTURE_RATE[] = {
02167     0.000, 23.976, 24.000, 25.000, 29.970, 30.000, 50.000, 59.940,
02168     60.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000
02169 };
02170
02171 static const uint8_t PLM_VIDEO_ZIG_ZAG[] = {
02172     0, 1, 8, 16, 9, 2, 3, 10,
02173     17, 24, 32, 25, 18, 11, 4, 5,
02174     12, 19, 26, 33, 40, 48, 41, 34,
02175     27, 20, 13, 6, 7, 14, 21, 28,
02176     35, 42, 49, 56, 57, 50, 43, 36,
02177     29, 22, 15, 23, 30, 37, 44, 51,
02178     58, 59, 52, 45, 38, 31, 39, 46,
02179     53, 60, 61, 54, 47, 55, 62, 63
02180 };
02181
02182 static const uint8_t PLM_VIDEO_INTRA_QUANT_MATRIX[] = {

```

```

02183     8, 16, 19, 22, 26, 27, 29, 34,
02184     16, 16, 22, 24, 27, 29, 34, 37,
02185     19, 22, 26, 27, 29, 34, 34, 38,
02186     22, 22, 26, 27, 29, 34, 37, 40,
02187     22, 26, 27, 29, 32, 35, 40, 48,
02188     26, 27, 29, 32, 35, 40, 48, 58,
02189     26, 27, 29, 34, 38, 46, 56, 69,
02190     27, 29, 35, 38, 46, 56, 69, 83
02191 };
02192
02193 static const uint8_t PLM_VIDEO_NON_INTRA_QUANT_MATRIX[] = {
02194     16, 16, 16, 16, 16, 16, 16, 16,
02195     16, 16, 16, 16, 16, 16, 16, 16,
02196     16, 16, 16, 16, 16, 16, 16, 16,
02197     16, 16, 16, 16, 16, 16, 16, 16,
02198     16, 16, 16, 16, 16, 16, 16, 16,
02199     16, 16, 16, 16, 16, 16, 16, 16,
02200     16, 16, 16, 16, 16, 16, 16, 16,
02201     16, 16, 16, 16, 16, 16, 16, 16
02202 };
02203
02204 static const uint8_t PLM_VIDEO_PREMULTIPLIER_MATRIX[] = {
02205     32, 44, 42, 38, 32, 25, 17, 9,
02206     44, 62, 58, 52, 44, 35, 24, 12,
02207     42, 58, 55, 49, 42, 33, 23, 12,
02208     38, 52, 49, 44, 38, 30, 20, 10,
02209     32, 44, 42, 38, 32, 25, 17, 9,
02210     25, 35, 33, 30, 25, 20, 14, 7,
02211     17, 24, 23, 20, 17, 14, 9, 5,
02212     9, 12, 12, 10, 9, 7, 5, 2
02213 };
02214
02215 static const plm_vlc_t PLM_VIDEO_MACROBLOCK_ADDRESS_INCREMENT[] = {
02216     { 1 < 1, 0}, { 0, 1}, // 0: x
02217     { 2 < 1, 0}, { 3 < 1, 0}, // 1: 0x
02218     { 4 < 1, 0}, { 5 < 1, 0}, // 2: 00x
02219     { 0, 3}, { 0, 2}, // 3: 01x
02220     { 6 < 1, 0}, { 7 < 1, 0}, // 4: 000x
02221     { 0, 5}, { 0, 4}, // 5: 001x
02222     { 8 < 1, 0}, { 9 < 1, 0}, // 6: 0000x
02223     { 0, 7}, { 0, 6}, // 7: 0001x
02224     { 10 < 1, 0}, { 11 < 1, 0}, // 8: 0000 0x
02225     { 12 < 1, 0}, { 13 < 1, 0}, // 9: 0000 1x
02226     { 14 < 1, 0}, { 15 < 1, 0}, // 10: 0000 00x
02227     { 16 < 1, 0}, { 17 < 1, 0}, // 11: 0000 01x
02228     { 18 < 1, 0}, { 19 < 1, 0}, // 12: 0000 10x
02229     { 0, 9}, { 0, 8}, // 13: 0000 11x
02230     { -1, 0}, { 20 < 1, 0}, // 14: 0000 000x
02231     { -1, 0}, { 21 < 1, 0}, // 15: 0000 001x
02232     { 22 < 1, 0}, { 23 < 1, 0}, // 16: 0000 010x
02233     { 0, 15}, { 0, 14}, // 17: 0000 011x
02234     { 0, 13}, { 0, 12}, // 18: 0000 100x
02235     { 0, 11}, { 0, 10}, // 19: 0000 101x
02236     { 24 < 1, 0}, { 25 < 1, 0}, // 20: 0000 0001x
02237     { 26 < 1, 0}, { 27 < 1, 0}, // 21: 0000 0011x
02238     { 28 < 1, 0}, { 29 < 1, 0}, // 22: 0000 0100x
02239     { 30 < 1, 0}, { 31 < 1, 0}, // 23: 0000 0101x
02240     { 32 < 1, 0}, { -1, 0}, // 24: 0000 0001 0x
02241     { -1, 0}, { 33 < 1, 0}, // 25: 0000 0001 1x
02242     { 34 < 1, 0}, { 35 < 1, 0}, // 26: 0000 0011 0x
02243     { 36 < 1, 0}, { 37 < 1, 0}, // 27: 0000 0011 1x
02244     { 38 < 1, 0}, { 39 < 1, 0}, // 28: 0000 0100 0x
02245     { 0, 21}, { 0, 20}, // 29: 0000 0100 1x
02246     { 0, 19}, { 0, 18}, // 30: 0000 0101 0x
02247     { 0, 17}, { 0, 16}, // 31: 0000 0101 1x
02248     { 0, 35}, { -1, 0}, // 32: 0000 0001 00x
02249     { -1, 0}, { 0, 34}, // 33: 0000 0001 11x
02250     { 0, 33}, { 0, 32}, // 34: 0000 0011 00x
02251     { 0, 31}, { 0, 30}, // 35: 0000 0011 01x
02252     { 0, 29}, { 0, 28}, // 36: 0000 0011 10x
02253     { 0, 27}, { 0, 26}, // 37: 0000 0011 11x
02254     { 0, 25}, { 0, 24}, // 38: 0000 0100 00x
02255     { 0, 23}, { 0, 22}, // 39: 0000 0100 01x
02256 };
02257
02258 static const plm_vlc_t PLM_VIDEO_MACROBLOCK_TYPE_INTRA[] = {
02259     { 1 < 1, 0}, { 0, 0x01}, // 0: x
02260     { -1, 0}, { 0, 0x11}, // 1: 0x
02261 };
02262
02263 static const plm_vlc_t PLM_VIDEO_MACROBLOCK_TYPE_PREDICTIVE[] = {
02264     { 1 < 1, 0}, { 0, 0x0a}, // 0: x
02265     { 2 < 1, 0}, { 0, 0x02}, // 1: 0x
02266     { 3 < 1, 0}, { 0, 0x08}, // 2: 00x
02267     { 4 < 1, 0}, { 5 < 1, 0}, // 3: 000x
02268     { 6 < 1, 0}, { 0, 0x12}, // 4: 0000x
02269     { 0, 0x1a}, { 0, 0x01}, // 5: 0001x

```

```

02270     {      -1,      0}, {      0, 0x11}, // 6: 0000 0x
02271 };
02272
02273 static const plm_vlc_t PLM_VIDEO_MACROBLOCK_TYPE_B[] = {
02274     { 1 << 1,      0}, { 2 << 1,      0}, // 0: x
02275     { 3 << 1,      0}, { 4 << 1,      0}, // 1: 0x
02276     { 0, 0x0c}, { 0, 0x0e}, // 2: 1x
02277     { 5 << 1,      0}, { 6 << 1,      0}, // 3: 00x
02278     { 0, 0x04}, { 0, 0x06}, // 4: 01x
02279     { 7 << 1,      0}, { 8 << 1,      0}, // 5: 000x
02280     { 0, 0x08}, { 0, 0x0a}, // 6: 001x
02281     { 9 << 1,      0}, {10 << 1,      0}, // 7: 0000x
02282     { 0, 0x1e}, { 0, 0x01}, // 8: 0001x
02283     {      -1,      0}, { 0, 0x11}, // 9: 0000 0x
02284     { 0, 0x16}, { 0, 0x1a}, // 10: 0000 1x
02285 };
02286
02287 static const plm_vlc_t *PLM_VIDEO_MACROBLOCK_TYPE[] = {
02288     NULL,
02289     PLM_VIDEO_MACROBLOCK_TYPE_INTRA,
02290     PLM_VIDEO_MACROBLOCK_TYPE_PREDICTIVE,
02291     PLM_VIDEO_MACROBLOCK_TYPE_B
02292 };
02293
02294 static const plm_vlc_t PLM_VIDEO_CODE_BLOCK_PATTERN[] = {
02295     { 1 << 1,      0}, { 2 << 1,      0}, // 0: x
02296     { 3 << 1,      0}, { 4 << 1,      0}, // 1: 0x
02297     { 5 << 1,      0}, { 6 << 1,      0}, // 2: 1x
02298     { 7 << 1,      0}, { 8 << 1,      0}, // 3: 00x
02299     { 9 << 1,      0}, {10 << 1,      0}, // 4: 01x
02300     {11 << 1,      0}, {12 << 1,      0}, // 5: 10x
02301     {13 << 1,      0}, { 0, 60}, // 6: 11x
02302     {14 << 1,      0}, {15 << 1,      0}, // 7: 000x
02303     {16 << 1,      0}, {17 << 1,      0}, // 8: 001x
02304     {18 << 1,      0}, {19 << 1,      0}, // 9: 010x
02305     {20 << 1,      0}, {21 << 1,      0}, // 10: 011x
02306     {22 << 1,      0}, {23 << 1,      0}, // 11: 100x
02307     { 0, 32}, { 0, 16}, // 12: 101x
02308     { 0, 8}, { 0, 4}, // 13: 110x
02309     {24 << 1,      0}, {25 << 1,      0}, // 14: 0000x
02310     {26 << 1,      0}, {27 << 1,      0}, // 15: 0001x
02311     {28 << 1,      0}, {29 << 1,      0}, // 16: 0010x
02312     {30 << 1,      0}, {31 << 1,      0}, // 17: 0011x
02313     { 0, 62}, { 0, 2}, // 18: 0100x
02314     { 0, 61}, { 0, 1}, // 19: 0101x
02315     { 0, 56}, { 0, 52}, // 20: 0110x
02316     { 0, 44}, { 0, 28}, // 21: 0111x
02317     { 0, 40}, { 0, 20}, // 22: 1000x
02318     { 0, 48}, { 0, 12}, // 23: 1001x
02319     {32 << 1,      0}, {33 << 1,      0}, // 24: 0000 0x
02320     {34 << 1,      0}, {35 << 1,      0}, // 25: 0000 1x
02321     {36 << 1,      0}, {37 << 1,      0}, // 26: 0001 0x
02322     {38 << 1,      0}, {39 << 1,      0}, // 27: 0001 1x
02323     {40 << 1,      0}, {41 << 1,      0}, // 28: 0010 0x
02324     {42 << 1,      0}, {43 << 1,      0}, // 29: 0010 1x
02325     { 0, 63}, { 0, 3}, // 30: 0011 0x
02326     { 0, 36}, { 0, 24}, // 31: 0011 1x
02327     {44 << 1,      0}, {45 << 1,      0}, // 32: 0000 00x
02328     {46 << 1,      0}, {47 << 1,      0}, // 33: 0000 01x
02329     {48 << 1,      0}, {49 << 1,      0}, // 34: 0000 10x
02330     {50 << 1,      0}, {51 << 1,      0}, // 35: 0000 11x
02331     {52 << 1,      0}, {53 << 1,      0}, // 36: 0001 00x
02332     {54 << 1,      0}, {55 << 1,      0}, // 37: 0001 01x
02333     {56 << 1,      0}, {57 << 1,      0}, // 38: 0001 10x
02334     {58 << 1,      0}, {59 << 1,      0}, // 39: 0001 11x
02335     { 0, 34}, { 0, 18}, // 40: 0010 00x
02336     { 0, 10}, { 0, 6}, // 41: 0010 01x
02337     { 0, 33}, { 0, 17}, // 42: 0010 10x
02338     { 0, 9}, { 0, 5}, // 43: 0010 11x
02339     {      -1,      0}, {60 << 1,      0}, // 44: 0000 000x
02340     {61 << 1,      0}, {62 << 1,      0}, // 45: 0000 001x
02341     { 0, 58}, { 0, 54}, // 46: 0000 010x
02342     { 0, 46}, { 0, 30}, // 47: 0000 011x
02343     { 0, 57}, { 0, 53}, // 48: 0000 100x
02344     { 0, 45}, { 0, 29}, // 49: 0000 101x
02345     { 0, 38}, { 0, 26}, // 50: 0000 110x
02346     { 0, 37}, { 0, 25}, // 51: 0000 111x
02347     { 0, 43}, { 0, 23}, // 52: 0001 000x
02348     { 0, 51}, { 0, 15}, // 53: 0001 001x
02349     { 0, 42}, { 0, 22}, // 54: 0001 010x
02350     { 0, 50}, { 0, 14}, // 55: 0001 011x
02351     { 0, 41}, { 0, 21}, // 56: 0001 100x
02352     { 0, 49}, { 0, 13}, // 57: 0001 101x
02353     { 0, 35}, { 0, 19}, // 58: 0001 110x
02354     { 0, 11}, { 0, 7}, // 59: 0001 111x
02355     { 0, 39}, { 0, 27}, // 60: 0000 0001x
02356     { 0, 59}, { 0, 55}, // 61: 0000 0010x

```

```

02357     {      0,    47}, {      0,    31}, // 62: 0000 0011x
02358 };
02359
02360 static const plm_vlc_t PLM_VIDEO_MOTION[] = {
02361     { 1 << 1,      0}, {      0,      0}, // 0: x
02362     { 2 << 1,      0}, { 3 << 1,      0}, // 1: 0x
02363     { 4 << 1,      0}, { 5 << 1,      0}, // 2: 00x
02364     {      0,      1}, {      0,   -1}, // 3: 01x
02365     { 6 << 1,      0}, { 7 << 1,      0}, // 4: 000x
02366     {      0,      2}, {      0,   -2}, // 5: 001x
02367     { 8 << 1,      0}, { 9 << 1,      0}, // 6: 0000x
02368     {      0,      3}, {      0,   -3}, // 7: 0001x
02369     { 10 << 1,     0}, { 11 << 1,     0}, // 8: 0000 0x
02370     { 12 << 1,     0}, { 13 << 1,     0}, // 9: 0000 1x
02371     {      -1,     0}, { 14 << 1,     0}, // 10: 0000 00x
02372     { 15 << 1,     0}, { 16 << 1,     0}, // 11: 0000 01x
02373     { 17 << 1,     0}, { 18 << 1,     0}, // 12: 0000 10x
02374     {      0,      4}, {      0,   -4}, // 13: 0000 11x
02375     {      -1,     0}, { 19 << 1,     0}, // 14: 0000 001x
02376     { 20 << 1,     0}, { 21 << 1,     0}, // 15: 0000 010x
02377     {      0,      7}, {      0,   -7}, // 16: 0000 011x
02378     {      0,      6}, {      0,   -6}, // 17: 0000 100x
02379     {      0,      5}, {      0,   -5}, // 18: 0000 101x
02380     { 22 << 1,     0}, { 23 << 1,     0}, // 19: 0000 0011x
02381     { 24 << 1,     0}, { 25 << 1,     0}, // 20: 0000 0100x
02382     { 26 << 1,     0}, { 27 << 1,     0}, // 21: 0000 0101x
02383     { 28 << 1,     0}, { 29 << 1,     0}, // 22: 0000 0011 0x
02384     { 30 << 1,     0}, { 31 << 1,     0}, // 23: 0000 0011 1x
02385     { 32 << 1,     0}, { 33 << 1,     0}, // 24: 0000 0100 0x
02386     {      0,     10}, {      0,   -10}, // 25: 0000 0100 1x
02387     {      0,     9}, {      0,   -9}, // 26: 0000 0101 0x
02388     {      0,     8}, {      0,   -8}, // 27: 0000 0101 1x
02389     {      0,    16}, {      0,   -16}, // 28: 0000 0011 00x
02390     {      0,    15}, {      0,   -15}, // 29: 0000 0011 01x
02391     {      0,    14}, {      0,   -14}, // 30: 0000 0011 10x
02392     {      0,    13}, {      0,   -13}, // 31: 0000 0011 11x
02393     {      0,    12}, {      0,   -12}, // 32: 0000 0100 00x
02394     {      0,    11}, {      0,   -11}, // 33: 0000 0100 01x
02395 };
02396
02397 static const plm_vlc_t PLM_VIDEO_DCT_SIZE_LUMINANCE[] = {
02398     { 1 << 1,      0}, { 2 << 1,      0}, // 0: x
02399     {      0,      1}, {      0,      2}, // 1: 0x
02400     { 3 << 1,      0}, { 4 << 1,      0}, // 2: 1x
02401     {      0,      0}, {      0,      3}, // 3: 10x
02402     {      0,      4}, { 5 << 1,      0}, // 4: 11x
02403     {      0,      5}, { 6 << 1,      0}, // 5: 111x
02404     {      0,      6}, { 7 << 1,      0}, // 6: 1111x
02405     {      0,      7}, { 8 << 1,      0}, // 7: 1111 1x
02406     {      0,      8}, {      -1,      0}, // 8: 1111 11x
02407 };
02408
02409 static const plm_vlc_t PLM_VIDEO_DCT_SIZE_CHROMINANCE[] = {
02410     { 1 << 1,      0}, { 2 << 1,      0}, // 0: x
02411     {      0,      0}, {      0,      1}, // 1: 0x
02412     {      0,      2}, { 3 << 1,      0}, // 2: 1x
02413     {      0,      3}, { 4 << 1,      0}, // 3: 11x
02414     {      0,      4}, { 5 << 1,      0}, // 4: 111x
02415     {      0,      5}, { 6 << 1,      0}, // 5: 1111x
02416     {      0,      6}, { 7 << 1,      0}, // 6: 1111 1x
02417     {      0,      7}, { 8 << 1,      0}, // 7: 1111 11x
02418     {      0,      8}, {      -1,      0}, // 8: 1111 111x
02419 };
02420
02421 static const plm_vlc_t *PLM_VIDEO_DCT_SIZE[] = {
02422     PLM_VIDEO_DCT_SIZE_LUMINANCE,
02423     PLM_VIDEO_DCT_SIZE_CHROMINANCE,
02424     PLM_VIDEO_DCT_SIZE_CHROMINANCE
02425 };
02426
02427
02428 // dct_coeff bitmap:
02429 // 0xff00 run
02430 // 0x00ff level
02431
02432 // Decoded values are unsigned. Sign bit follows in the stream.
02433
02434 static const plm_vlc_uint_t PLM_VIDEO_DCT_COEFF[] = {
02435     { 1 << 1,      0}, {      0, 0x0001}, // 0: x
02436     { 2 << 1,      0}, { 3 << 1,      0}, // 1: 0x
02437     { 4 << 1,      0}, { 5 << 1,      0}, // 2: 00x
02438     { 6 << 1,      0}, {      0, 0x0101}, // 3: 01x
02439     { 7 << 1,      0}, { 8 << 1,      0}, // 4: 000x
02440     { 9 << 1,      0}, {10 << 1,      0}, // 5: 001x
02441     {      0, 0x0002}, {      0, 0x0201}, // 6: 010x
02442     {11 << 1,      0}, {12 << 1,      0}, // 7: 0000x
02443     {13 << 1,      0}, {14 << 1,      0}, // 8: 0001x

```

```

02444 { 15 < 1,          0}, {      0, 0x0003}, //  9: 0010x
02445 {      0, 0x0401}, {      0, 0x0301}, // 10: 0011x
02446 { 16 < 1,          0}, {      0, 0xfffff}, // 11: 0000 0x
02447 { 17 < 1,          0}, { 18 < 1,      0}, // 12: 0000 1x
02448 {      0, 0x0701}, {      0, 0x0601}, // 13: 0001 0x
02449 {      0, 0x0102}, {      0, 0x0501}, // 14: 0001 1x
02450 { 19 < 1,          0}, { 20 < 1,      0}, // 15: 0010 0x
02451 { 21 < 1,          0}, { 22 < 1,      0}, // 16: 0000 00x
02452 {      0, 0x0202}, {      0, 0x0901}, // 17: 0000 10x
02453 {      0, 0x0004}, {      0, 0x0801}, // 18: 0000 11x
02454 { 23 < 1,          0}, { 24 < 1,      0}, // 19: 0010 00x
02455 { 25 < 1,          0}, { 26 < 1,      0}, // 20: 0010 01x
02456 { 27 < 1,          0}, { 28 < 1,      0}, // 21: 0000 000x
02457 { 29 < 1,          0}, { 30 < 1,      0}, // 22: 0000 001x
02458 {      0, 0x0d01}, {      0, 0x0006}, // 23: 0010 000x
02459 {      0, 0x0c01}, {      0, 0x0b01}, // 24: 0010 001x
02460 {      0, 0x0302}, {      0, 0x0103}, // 25: 0010 010x
02461 {      0, 0x0005}, {      0, 0x0a01}, // 26: 0010 011x
02462 { 31 < 1,          0}, { 32 < 1,      0}, // 27: 0000 0000x
02463 { 33 < 1,          0}, { 34 < 1,      0}, // 28: 0000 0001x
02464 { 35 < 1,          0}, { 36 < 1,      0}, // 29: 0000 0010x
02465 { 37 < 1,          0}, { 38 < 1,      0}, // 30: 0000 0011x
02466 { 39 < 1,          0}, { 40 < 1,      0}, // 31: 0000 0000 0x
02467 { 41 < 1,          0}, { 42 < 1,      0}, // 32: 0000 0000 1x
02468 { 43 < 1,          0}, { 44 < 1,      0}, // 33: 0000 0001 0x
02469 { 45 < 1,          0}, { 46 < 1,      0}, // 34: 0000 0001 1x
02470 {      0, 0x1001}, {      0, 0x0502}, // 35: 0000 0010 0x
02471 {      0, 0x0007}, {      0, 0x0203}, // 36: 0000 0010 1x
02472 {      0, 0x0104}, {      0, 0x0f01}, // 37: 0000 0011 0x
02473 {      0, 0x0e01}, {      0, 0x0402}, // 38: 0000 0011 1x
02474 { 47 < 1,          0}, { 48 < 1,      0}, // 39: 0000 0000 00x
02475 { 49 < 1,          0}, { 50 < 1,      0}, // 40: 0000 0000 01x
02476 { 51 < 1,          0}, { 52 < 1,      0}, // 41: 0000 0000 10x
02477 { 53 < 1,          0}, { 54 < 1,      0}, // 42: 0000 0000 11x
02478 { 55 < 1,          0}, { 56 < 1,      0}, // 43: 0000 0001 00x
02479 { 57 < 1,          0}, { 58 < 1,      0}, // 44: 0000 0001 01x
02480 { 59 < 1,          0}, { 60 < 1,      0}, // 45: 0000 0001 10x
02481 { 61 < 1,          0}, { 62 < 1,      0}, // 46: 0000 0001 11x
02482 {     -1,          0}, { 63 < 1,      0}, // 47: 0000 0000 000x
02483 { 64 < 1,          0}, { 65 < 1,      0}, // 48: 0000 0000 001x
02484 { 66 < 1,          0}, { 67 < 1,      0}, // 49: 0000 0000 010x
02485 { 68 < 1,          0}, { 69 < 1,      0}, // 50: 0000 0000 011x
02486 { 70 < 1,          0}, { 71 < 1,      0}, // 51: 0000 0000 100x
02487 { 72 < 1,          0}, { 73 < 1,      0}, // 52: 0000 0000 101x
02488 { 74 < 1,          0}, { 75 < 1,      0}, // 53: 0000 0000 110x
02489 { 76 < 1,          0}, { 77 < 1,      0}, // 54: 0000 0000 111x
02490 {      0, 0x000b}, {      0, 0x0802}, // 55: 0000 0001 000x
02491 {      0, 0x0403}, {      0, 0x000a}, // 56: 0000 0001 001x
02492 {      0, 0x0204}, {      0, 0x0702}, // 57: 0000 0001 010x
02493 {      0, 0x1501}, {      0, 0x1401}, // 58: 0000 0001 011x
02494 {      0, 0x0009}, {      0, 0x1301}, // 59: 0000 0001 100x
02495 {      0, 0x1201}, {      0, 0x0105}, // 60: 0000 0001 101x
02496 {      0, 0x0303}, {      0, 0x0008}, // 61: 0000 0001 110x
02497 {      0, 0x0602}, {      0, 0x1101}, // 62: 0000 0001 111x
02498 { 78 < 1,          0}, { 79 < 1,      0}, // 63: 0000 0000 0001x
02499 { 80 < 1,          0}, { 81 < 1,      0}, // 64: 0000 0000 0010x
02500 { 82 < 1,          0}, { 83 < 1,      0}, // 65: 0000 0000 0011x
02501 { 84 < 1,          0}, { 85 < 1,      0}, // 66: 0000 0000 0100x
02502 { 86 < 1,          0}, { 87 < 1,      0}, // 67: 0000 0000 0101x
02503 { 88 < 1,          0}, { 89 < 1,      0}, // 68: 0000 0000 0110x
02504 { 90 < 1,          0}, { 91 < 1,      0}, // 69: 0000 0000 0111x
02505 {      0, 0x0a02}, {      0, 0x0902}, // 70: 0000 0000 1000x
02506 {      0, 0x0503}, {      0, 0x0304}, // 71: 0000 0000 1001x
02507 {      0, 0x0205}, {      0, 0x0107}, // 72: 0000 0000 1010x
02508 {      0, 0x0106}, {      0, 0x000f}, // 73: 0000 0000 1011x
02509 {      0, 0x000e}, {      0, 0x000d}, // 74: 0000 0000 1100x
02510 {      0, 0x000c}, {      0, 0x1a01}, // 75: 0000 0000 1101x
02511 {      0, 0x1901}, {      0, 0x1801}, // 76: 0000 0000 1110x
02512 {      0, 0x1701}, {      0, 0x1601}, // 77: 0000 0000 1111x
02513 { 92 < 1,          0}, { 93 < 1,      0}, // 78: 0000 0000 0001 0x
02514 { 94 < 1,          0}, { 95 < 1,      0}, // 79: 0000 0000 0001 1x
02515 { 96 < 1,          0}, { 97 < 1,      0}, // 80: 0000 0000 0010 0x
02516 { 98 < 1,          0}, { 99 < 1,      0}, // 81: 0000 0000 0010 1x
02517 {100 < 1,          0}, {101 < 1,      0}, // 82: 0000 0000 0011 0x
02518 {102 < 1,          0}, {103 < 1,      0}, // 83: 0000 0000 0011 1x
02519 {      0, 0x001f}, {      0, 0x001e}, // 84: 0000 0000 0100 0x
02520 {      0, 0x001d}, {      0, 0x001c}, // 85: 0000 0000 0100 1x
02521 {      0, 0x001b}, {      0, 0x001a}, // 86: 0000 0000 0101 0x
02522 {      0, 0x0019}, {      0, 0x0018}, // 87: 0000 0000 0101 1x
02523 {      0, 0x0017}, {      0, 0x0016}, // 88: 0000 0000 0110 0x
02524 {      0, 0x0015}, {      0, 0x0014}, // 89: 0000 0000 0110 1x
02525 {      0, 0x0013}, {      0, 0x0012}, // 90: 0000 0000 0111 0x
02526 {      0, 0x0011}, {      0, 0x0010}, // 91: 0000 0000 0111 1x
02527 {104 < 1,          0}, {105 < 1,      0}, // 92: 0000 0000 0001 00x
02528 {106 < 1,          0}, {107 < 1,      0}, // 93: 0000 0000 0001 01x
02529 {108 < 1,          0}, {109 < 1,      0}, // 94: 0000 0000 0001 10x
02530 {110 < 1,          0}, {111 < 1,      0}, // 95: 0000 0000 0001 11x

```

```

02531     {      0, 0x0028}, {      0, 0x0027}, // 96: 0000 0000 0010 00x
02532     {      0, 0x0026}, {      0, 0x0025}, // 97: 0000 0000 0010 01x
02533     {      0, 0x0024}, {      0, 0x0023}, // 98: 0000 0000 0010 10x
02534     {      0, 0x0022}, {      0, 0x0021}, // 99: 0000 0000 0010 11x
02535     {      0, 0x0020}, {      0, 0x010e}, // 100: 0000 0000 0011 00x
02536     {      0, 0x010d}, {      0, 0x010c}, // 101: 0000 0000 0011 01x
02537     {      0, 0x010b}, {      0, 0x010a}, // 102: 0000 0000 0011 10x
02538     {      0, 0x0109}, {      0, 0x0108}, // 103: 0000 0000 0011 11x
02539     {      0, 0x0112}, {      0, 0x0111}, // 104: 0000 0000 0001 000x
02540     {      0, 0x0110}, {      0, 0x010f}, // 105: 0000 0000 0001 001x
02541     {      0, 0x0603}, {      0, 0x1002}, // 106: 0000 0000 0001 010x
02542     {      0, 0x0f02}, {      0, 0x0e02}, // 107: 0000 0000 0001 011x
02543     {      0, 0x0d02}, {      0, 0x0c02}, // 108: 0000 0000 0001 100x
02544     {      0, 0x0b02}, {      0, 0x1f01}, // 109: 0000 0000 0001 101x
02545     {      0, 0x1e01}, {      0, 0x1d01}, // 110: 0000 0000 0001 110x
02546     {      0, 0x1c01}, {      0, 0x1b01}, // 111: 0000 0000 0001 111x
02547 };
02548
02549 typedef struct {
02550     int full_px;
02551     int is_set;
02552     int r_size;
02553     int h;
02554     int v;
02555 } plm_video_motion_t;
02556
02557 struct plm_video_t {
02558     double framerate;
02559     double time;
02560     int frames_decoded;
02561     int width;
02562     int height;
02563     int mb_width;
02564     int mb_height;
02565     int mb_size;
02566
02567     int luma_width;
02568     int luma_height;
02569
02570     int chroma_width;
02571     int chroma_height;
02572
02573     int start_code;
02574     int picture_type;
02575
02576     plm_video_motion_t motion_forward;
02577     plm_video_motion_t motion_backward;
02578
02579     int has_sequence_header;
02580
02581     int quantizer_scale;
02582     int slice_begin;
02583     int macroblock_address;
02584
02585     int mb_row;
02586     int mb_col;
02587
02588     int macroblock_type;
02589     int macroblock_intra;
02590
02591     int dc_predictor[3];
02592
02593     plm_buffer_t *buffer;
02594     int destroy_buffer_when_done;
02595
02596     plm_frame_t frame_current;
02597     plm_frame_t frame_forward;
02598     plm_frame_t frame_backward;
02599
02600     uint8_t *frames_data;
02601
02602     int block_data[64];
02603     uint8_t intra_quant_matrix[64];
02604     uint8_t non_intra_quant_matrix[64];
02605
02606     int has_reference_frame;
02607     int assume_no_b_frames;
02608 };
02609
02610 static inline uint8_t plm_clamp(int n) {
02611     if (n > 255) {
02612         n = 255;
02613     }
02614     else if (n < 0) {
02615         n = 0;
02616     }
02617     return n;

```

```
02618 }
02619
02620 int plm_video_decode_sequence_header(plm_video_t *self);
02621 void plm_video_init_frame(plm_video_t *self, plm_frame_t *frame, uint8_t *base);
02622 void plm_video_decode_picture(plm_video_t *self);
02623 void plm_video_decode_slice(plm_video_t *self, int slice);
02624 void plm_video_decode_macroblock(plm_video_t *self);
02625 void plm_video_decode_motion_vectors(plm_video_t *self);
02626 int plm_video_decode_motion_vector(plm_video_t *self, int r_size, int motion);
02627 void plm_video_predict_macroblock(plm_video_t *self);
02628 void plm_video_copy_macroblock(plm_video_t *self, plm_frame_t *s, int motion_h, int motion_v);
02629 void plm_video_interpolate_macroblock(plm_video_t *self, plm_frame_t *s, int motion_h, int motion_v);
02630 void plm_video_process_macroblock(plm_video_t *self, uint8_t *s, uint8_t *d, int mh, int mb, int bs,
02631     int interp);
02632 void plm_video_decode_block(plm_video_t *self, int block);
02633 void plm_video_idct(int *block);
02634
02635 plm_video_t * plm_video_create_with_buffer(plm_buffer_t *buffer, int destroy_when_done) {
02636     plm_video_t *self = (plm_video_t *)PLM_MALLOC(sizeof(plm_video_t));
02637     memset(self, 0, sizeof(plm_video_t));
02638
02639     self->buffer = buffer;
02640     self->destroy_buffer_when_done = destroy_when_done;
02641
02642     // Attempt to decode the sequence header
02643     self->start_code = plm_buffer_find_start_code(self->buffer, PLM_START_SEQUENCE);
02644     if (self->start_code != -1) {
02645         plm_video_decode_sequence_header(self);
02646     }
02647     return self;
02648 }
02649 void plm_video_destroy(plm_video_t *self) {
02650     if (self->destroy_buffer_when_done) {
02651         plm_buffer_destroy(self->buffer);
02652     }
02653
02654     if (self->has_sequence_header) {
02655         PLM_FREE(self->frames_data);
02656     }
02657
02658     PLM_FREE(self);
02659 }
02660
02661 double plm_video_get_framerate(plm_video_t *self) {
02662     return plm_video_has_header(self)
02663         ? self->framerate
02664         : 0;
02665 }
02666
02667 int plm_video_get_width(plm_video_t *self) {
02668     return plm_video_has_header(self)
02669         ? self->width
02670         : 0;
02671 }
02672
02673 int plm_video_get_height(plm_video_t *self) {
02674     return plm_video_has_header(self)
02675         ? self->height
02676         : 0;
02677 }
02678
02679 void plm_video_set_no_delay(plm_video_t *self, int no_delay) {
02680     self->assume_no_b_frames = no_delay;
02681 }
02682
02683 double plm_video_get_time(plm_video_t *self) {
02684     return self->time;
02685 }
02686
02687 void plm_video_set_time(plm_video_t *self, double time) {
02688     self->frames_decoded = self->framerate * time;
02689     self->time = time;
02690 }
02691
02692 void plm_video_rewind(plm_video_t *self) {
02693     plm_buffer_rewind(self->buffer);
02694     self->time = 0;
02695     self->frames_decoded = 0;
02696     self->has_reference_frame = FALSE;
02697     self->start_code = -1;
02698 }
02699
02700 int plm_video_hasEnded(plm_video_t *self) {
02701     return plm_buffer_hasEnded(self->buffer);
02702 }
02703
```

```

02704 plm_frame_t *plm_video_decode(plm_video_t *self) {
02705     if (!plm_video_has_header(self)) {
02706         return NULL;
02707     }
02708
02709     plm_frame_t *frame = NULL;
02710     do {
02711         if (self->start_code != PLM_START_PICTURE) {
02712             self->start_code = plm_buffer_find_start_code(self->buffer, PLM_START_PICTURE);
02713
02714             if (self->start_code == -1) {
02715                 // If we reached the end of the file and the previously decoded
02716                 // frame was a reference frame, we still have to return it.
02717                 if (
02718                     self->has_reference_frame &&
02719                     !self->assume_no_b_frames &&
02720                     plm_buffer_has-ended(self->buffer) &&
02721                     self->picture_type == PLM_VIDEO_PICTURE_TYPE_INTRA || 
02722                     self->picture_type == PLM_VIDEO_PICTURE_TYPE_PREDICTIVE
02723                 )
02724             ) {
02725                 self->has_reference_frame = FALSE;
02726                 frame = &self->frame_backward;
02727                 break;
02728             }
02729
02730             return NULL;
02731         }
02732     }
02733
02734     // Make sure we have a full picture in the buffer before attempting to
02735     // decode it. Sadly, this can only be done by seeking for the start code
02736     // of the next picture. Also, if we didn't find the start code for the
02737     // next picture, but the source has ended, we assume that this last
02738     // picture is in the buffer.
02739     if (
02740         plm_buffer_has_start_code(self->buffer, PLM_START_PICTURE) == -1 &&
02741         !plm_buffer_has-ended(self->buffer)
02742     ) {
02743         return NULL;
02744     }
02745     plm_buffer_discard_read_bytes(self->buffer);
02746
02747     plm_video_decode_picture(self);
02748
02749     if (self->assume_no_b_frames) {
02750         frame = &self->frame_backward;
02751     }
02752     else if (self->picture_type == PLM_VIDEO_PICTURE_TYPE_B) {
02753         frame = &self->frame_current;
02754     }
02755     else if (self->has_reference_frame) {
02756         frame = &self->frame_forward;
02757     }
02758     else {
02759         self->has_reference_frame = TRUE;
02760     }
02761 } while (!frame);
02762
02763 frame->time = self->time;
02764 self->frames_decoded++;
02765 self->time = (double)self->frames_decoded / self->framerate;
02766
02767 return frame;
02768 }
02769
02770 int plm_video_has_header(plm_video_t *self) {
02771     if (self->has_sequence_header) {
02772         return TRUE;
02773     }
02774
02775     if (self->start_code != PLM_START_SEQUENCE) {
02776         self->start_code = plm_buffer_find_start_code(self->buffer, PLM_START_SEQUENCE);
02777     }
02778     if (self->start_code == -1) {
02779         return FALSE;
02780     }
02781
02782     if (!plm_video_decode_sequence_header(self)) {
02783         return FALSE;
02784     }
02785
02786     return TRUE;
02787 }
02788
02789 int plm_video_decode_sequence_header(plm_video_t *self) {
02790     int max_header_size = 64 + 2 * 64 * 8; // 64 bit header + 2x 64 byte matrix

```

```

02791     if (!plm_buffer_has(self->buffer, max_header_size)) {
02792         return FALSE;
02793     }
02794
02795     self->width = plm_buffer_read(self->buffer, 12);
02796     self->height = plm_buffer_read(self->buffer, 12);
02797
02798     if (self->width <= 0 || self->height <= 0) {
02799         return FALSE;
02800     }
02801
02802     // Skip pixel aspect ratio
02803     plm_buffer_skip(self->buffer, 4);
02804
02805     self->framerate = PLM_VIDEO_PICTURE_RATE[plm_buffer_read(self->buffer, 4)];
02806
02807     // Skip bit_rate, marker, buffer_size and constrained bit
02808     plm_buffer_skip(self->buffer, 18 + 1 + 10 + 1);
02809
02810     // Load custom intra quant matrix?
02811     if (plm_buffer_read(self->buffer, 1)) {
02812         for (int i = 0; i < 64; i++) {
02813             int idx = PLM_VIDEO_ZIG_ZAG[i];
02814             self->intra_quant_matrix[idx] = plm_buffer_read(self->buffer, 8);
02815         }
02816     }
02817     else {
02818         memcpy(self->intra_quant_matrix, PLM_VIDEO_INTRA_QUANT_MATRIX, 64);
02819     }
02820
02821     // Load custom non intra quant matrix?
02822     if (plm_buffer_read(self->buffer, 1)) {
02823         for (int i = 0; i < 64; i++) {
02824             int idx = PLM_VIDEO_ZIG_ZAG[i];
02825             self->non_intra_quant_matrix[idx] = plm_buffer_read(self->buffer, 8);
02826         }
02827     }
02828     else {
02829         memcpy(self->non_intra_quant_matrix, PLM_VIDEO_NON_INTRA_QUANT_MATRIX, 64);
02830     }
02831
02832     self->mb_width = (self->width + 15) >> 4;
02833     self->mb_height = (self->height + 15) >> 4;
02834     self->mb_size = self->mb_width * self->mb_height;
02835
02836     self->luma_width = self->mb_width << 4;
02837     self->luma_height = self->mb_height << 4;
02838
02839     self->chroma_width = self->mb_width << 3;
02840     self->chroma_height = self->mb_height << 3;
02841
02842
02843     // Allocate one big chunk of data for all 3 frames = 9 planes
02844     size_t luma_plane_size = self->luma_width * self->luma_height;
02845     size_t chroma_plane_size = self->chroma_width * self->chroma_height;
02846     size_t frame_data_size = (luma_plane_size + 2 * chroma_plane_size);
02847
02848     self->frames_data = (uint8_t*)PLM_MALLOC(frame_data_size * 3);
02849     plm_video_init_frame(self, &self->frame_current, self->frames_data + frame_data_size * 0);
02850     plm_video_init_frame(self, &self->frame_forward, self->frames_data + frame_data_size * 1);
02851     plm_video_init_frame(self, &self->frame_backward, self->frames_data + frame_data_size * 2);
02852
02853     self->has_sequence_header = TRUE;
02854     return TRUE;
02855 }
02856
02857 void plm_video_init_frame(plm_video_t *self, plm_frame_t *frame, uint8_t *base) {
02858     size_t luma_plane_size = self->luma_width * self->luma_height;
02859     size_t chroma_plane_size = self->chroma_width * self->chroma_height;
02860
02861     frame->width = self->width;
02862     frame->height = self->height;
02863     frame->y.width = self->luma_width;
02864     frame->y.height = self->luma_height;
02865     frame->y.data = base;
02866
02867     frame->cr.width = self->chroma_width;
02868     frame->cr.height = self->chroma_height;
02869     frame->cr.data = base + luma_plane_size;
02870
02871     frame->cb.width = self->chroma_width;
02872     frame->cb.height = self->chroma_height;
02873     frame->cb.data = base + luma_plane_size + chroma_plane_size;
02874 }
02875
02876 void plm_video_decode_picture(plm_video_t *self) {
02877     plm_buffer_skip(self->buffer, 10); // skip temporalReference

```

```

02878     self->picture_type = plm_buffer_read(self->buffer, 3);
02879     plm_buffer_skip(self->buffer, 16); // skip vbv_delay
02880
02881 // D frames or unknown coding type
02882 if (self->picture_type <= 0 || self->picture_type > PLM_VIDEO_PICTURE_TYPE_B) {
02883     return;
02884 }
02885
02886 // Forward full_px, f_code
02887 if (
02888     self->picture_type == PLM_VIDEO_PICTURE_TYPE_PREDICTIVE ||
02889     self->picture_type == PLM_VIDEO_PICTURE_TYPE_B
02890 ) {
02891     self->motion_forward.full_px = plm_buffer_read(self->buffer, 1);
02892     int f_code = plm_buffer_read(self->buffer, 3);
02893     if (f_code == 0) {
02894         // Ignore picture with zero f_code
02895         return;
02896     }
02897     self->motion_forward.r_size = f_code - 1;
02898 }
02899
02900 // Backward full_px, f_code
02901 if (self->picture_type == PLM_VIDEO_PICTURE_TYPE_B) {
02902     self->motion_backward.full_px = plm_buffer_read(self->buffer, 1);
02903     int f_code = plm_buffer_read(self->buffer, 3);
02904     if (f_code == 0) {
02905         // Ignore picture with zero f_code
02906         return;
02907     }
02908     self->motion_backward.r_size = f_code - 1;
02909 }
02910
02911 plm_frame_t frame_temp = self->frame_forward;
02912 if (
02913     self->picture_type == PLM_VIDEO_PICTURE_TYPE_INTRA ||
02914     self->picture_type == PLM_VIDEO_PICTURE_TYPE_PREDICTIVE
02915 ) {
02916     self->frame_forward = self->frame_backward;
02917 }
02918
02919
02920 // Find first slice start code; skip extension and user data
02921 do {
02922     self->start_code = plm_buffer_next_start_code(self->buffer);
02923 } while (
02924     self->start_code == PLM_START_EXTENSION ||
02925     self->start_code == PLM_START_USER_DATA
02926 );
02927
02928 // Decode all slices
02929 while (PLM_START_IS_SLICE(self->start_code)) {
02930     plm_video_decode_slice(self, self->start_code & 0x000000FF);
02931     if (self->macroblock_address >= self->mb_size - 2) {
02932         break;
02933     }
02934     self->start_code = plm_buffer_next_start_code(self->buffer);
02935 }
02936
02937 // If this is a reference picture rotate the prediction pointers
02938 if (
02939     self->picture_type == PLM_VIDEO_PICTURE_TYPE_INTRA ||
02940     self->picture_type == PLM_VIDEO_PICTURE_TYPE_PREDICTIVE
02941 ) {
02942     self->frame_backward = self->frame_current;
02943     self->frame_current = frame_temp;
02944 }
02945 }
02946
02947 void plm_video_decode_slice(plm_video_t *self, int slice) {
02948     self->slice_begin = TRUE;
02949     self->macroblock_address = (slice - 1) * self->mb_width - 1;
02950
02951 // Reset motion vectors and DC predictors
02952     self->motion_backward.h = self->motion_forward.h = 0;
02953     self->motion_backward.v = self->motion_forward.v = 0;
02954     self->dc_predictor[0] = 128;
02955     self->dc_predictor[1] = 128;
02956     self->dc_predictor[2] = 128;
02957
02958     self->quantizer_scale = plm_buffer_read(self->buffer, 5);
02959
02960 // Skip extra
02961 while (plm_buffer_read(self->buffer, 1)) {
02962     plm_buffer_skip(self->buffer, 8);
02963 }
02964

```

```

02965     do {
02966         plm_video_decode_macroblock(self);
02967     } while (
02968         self->macroblock_address < self->mb_size - 1 &&
02969         plm_buffer_peek_non_zero(self->buffer, 23)
02970     );
02971 }
02972
02973 void plm_video_decode_macroblock(plm_video_t *self) {
02974     // Decode increment
02975     int increment = 0;
02976     int t = plm_buffer_read_vlc(self->buffer, PLM_VIDEO_MACROBLOCK_ADDRESS_INCREMENT);
02977
02978     while (t == 34) {
02979         // macroblock_stuffing
02980         t = plm_buffer_read_vlc(self->buffer, PLM_VIDEO_MACROBLOCK_ADDRESS_INCREMENT);
02981     }
02982     while (t == 35) {
02983         // macroblock_escape
02984         increment += 33;
02985         t = plm_buffer_read_vlc(self->buffer, PLM_VIDEO_MACROBLOCK_ADDRESS_INCREMENT);
02986     }
02987     increment += t;
02988
02989     // Process any skipped macroblocks
02990     if (self->slice_begin) {
02991         // The first increment of each slice is relative to beginning of the
02992         // previous row, not the previous macroblock
02993         self->slice_begin = FALSE;
02994         self->macroblock_address += increment;
02995     }
02996     else {
02997         if (self->macroblock_address + increment >= self->mb_size) {
02998             return; // invalid
02999         }
03000         if (increment > 1) {
03001             // Skipped macroblocks reset DC predictors
03002             self->dc_predictor[0] = 128;
03003             self->dc_predictor[1] = 128;
03004             self->dc_predictor[2] = 128;
03005
03006             // Skipped macroblocks in P-pictures reset motion vectors
03007             if (self->picture_type == PLM_VIDEO_PICTURE_TYPE_PREDICTIVE) {
03008                 self->motion_forward.h = 0;
03009                 self->motion_forward.v = 0;
03010             }
03011         }
03012
03013         // Predict skipped macroblocks
03014         while (increment > 1) {
03015             self->macroblock_address++;
03016             self->mb_row = self->macroblock_address / self->mb_width;
03017             self->mb_col = self->macroblock_address % self->mb_width;
03018
03019             plm_video_predict_macroblock(self);
03020             increment--;
03021         }
03022         self->macroblock_address++;
03023     }
03024
03025     self->mb_row = self->macroblock_address / self->mb_width;
03026     self->mb_col = self->macroblock_address % self->mb_width;
03027
03028     if (self->mb_col >= self->mb_width || self->mb_row >= self->mb_height) {
03029         return; // corrupt stream;
03030     }
03031
03032     // Process the current macroblock
03033     const plm_vlc_t *table = PLM_VIDEO_MACROBLOCK_TYPE[self->picture_type];
03034     self->macroblock_type = plm_buffer_read_vlc(self->buffer, table);
03035
03036     self->macroblock_intra = (self->macroblock_type & 0x01);
03037     self->motion_forward.is_set = (self->macroblock_type & 0x08);
03038     self->motion_backward.is_set = (self->macroblock_type & 0x04);
03039
03040     // Quantizer scale
03041     if (((self->macroblock_type & 0x10) != 0) {
03042         self->quantizer_scale = plm_buffer_read(self->buffer, 5);
03043     }
03044
03045     if (self->macroblock_intra) {
03046         // Intra-coded macroblocks reset motion vectors
03047         self->motion_backward.h = self->motion_forward.h = 0;
03048         self->motion_backward.v = self->motion_forward.v = 0;
03049     }
03050     else {
03051         // Non-intra macroblocks reset DC predictors

```

```

03052         self->dc_predictor[0] = 128;
03053         self->dc_predictor[1] = 128;
03054         self->dc_predictor[2] = 128;
03055
03056         plm_video_decode_motion_vectors(self);
03057         plm_video_predict_macroblock(self);
03058     }
03059
03060     // Decode blocks
03061     int cbp = ((self->macroblock_type & 0x02) != 0)
03062         ? plm_buffer_read_vlc(self->buffer, PLM_VIDEO_CODE_BLOCK_PATTERN)
03063         : (self->macroblock_intra ? 0x3f : 0);
03064
03065     for (int block = 0, mask = 0x20; block < 6; block++) {
03066         if ((cbp & mask) != 0) {
03067             plm_video_decode_block(self, block);
03068         }
03069         mask >>= 1;
03070     }
03071 }
03072
03073 void plm_video_decode_motion_vectors(plm_video_t *self) {
03074
03075     // Forward
03076     if (self->motion_forward.is_set) {
03077         int r_size = self->motion_forward.r_size;
03078         self->motion_forward.h = plm_video_decode_motion_vector(self, r_size, self->motion_forward.h);
03079         self->motion_forward.v = plm_video_decode_motion_vector(self, r_size, self->motion_forward.v);
03080     }
03081     else if (self->picture_type == PLM_VIDEO_PICTURE_TYPE_PREDICTIVE) {
03082         // No motion information in P-picture, reset vectors
03083         self->motion_forward.h = 0;
03084         self->motion_forward.v = 0;
03085     }
03086
03087     if (self->motion_backward.is_set) {
03088         int r_size = self->motion_backward.r_size;
03089         self->motion_backward.h = plm_video_decode_motion_vector(self, r_size,
03090             self->motion_backward.h);
03090         self->motion_backward.v = plm_video_decode_motion_vector(self, r_size,
03091             self->motion_backward.v);
03092     }
03093
03094 int plm_video_decode_motion_vector(plm_video_t *self, int r_size, int motion) {
03095     int fscale = 1 << r_size;
03096     int m_code = plm_buffer_read_vlc(self->buffer, PLM_VIDEO_MOTION);
03097     int r = 0;
03098     int d;
03099
03100     if ((m_code != 0) && (fscale != 1)) {
03101         r = plm_buffer_read(self->buffer, r_size);
03102         d = ((abs(m_code) - 1) << r_size) + r + 1;
03103         if (m_code < 0) {
03104             d = -d;
03105         }
03106     }
03107     else {
03108         d = m_code;
03109     }
03110
03111     motion += d;
03112     if (motion > (fscale << 4) - 1) {
03113         motion -= fscale << 5;
03114     }
03115     else if (motion < ((-fscale) << 4)) {
03116         motion += fscale << 5;
03117     }
03118
03119     return motion;
03120 }
03121
03122 void plm_video_predict_macroblock(plm_video_t *self) {
03123     int fw_h = self->motion_forward.h;
03124     int fw_v = self->motion_forward.v;
03125
03126     if (self->motion_forward.full_px) {
03127         fw_h <<= 1;
03128         fw_v <<= 1;
03129     }
03130
03131     if (self->picture_type == PLM_VIDEO_PICTURE_TYPE_B) {
03132         int bw_h = self->motion_backward.h;
03133         int bw_v = self->motion_backward.v;
03134
03135         if (self->motion_backward.full_px) {
03136             bw_h <<= 1;

```

```

03137         bw_v <= 1;
03138     }
03139
03140     if (self->motion_forward.is_set) {
03141         plm_video_copy_macroblock(self, &self->frame_forward, fw_h, fw_v);
03142         if (self->motion_backward.is_set) {
03143             plm_video_interpolate_macroblock(self, &self->frame_backward, bw_h, bw_v);
03144         }
03145     }
03146     else {
03147         plm_video_copy_macroblock(self, &self->frame_backward, bw_h, bw_v);
03148     }
03149 }
03150 else {
03151     plm_video_copy_macroblock(self, &self->frame_forward, fw_h, fw_v);
03152 }
03153 }
03154
03155 void plm_video_copy_macroblock(plm_video_t *self, plm_frame_t *s, int motion_h, int motion_v) {
03156     plm_frame_t *d = &self->frame_current;
03157     plm_video_process_macroblock(self, s->y.data, d->y.data, motion_h, motion_v, 16, FALSE);
03158     plm_video_process_macroblock(self, s->cr.data, d->cr.data, motion_h / 2, motion_v / 2, 8, FALSE);
03159     plm_video_process_macroblock(self, s->cb.data, d->cb.data, motion_h / 2, motion_v / 2, 8, FALSE);
03160 }
03161
03162 void plm_video_interpolate_macroblock(plm_video_t *self, plm_frame_t *s, int motion_h, int motion_v) {
03163     plm_frame_t *d = &self->frame_current;
03164     plm_video_process_macroblock(self, s->y.data, d->y.data, motion_h, motion_v, 16, TRUE);
03165     plm_video_process_macroblock(self, s->cr.data, d->cr.data, motion_h / 2, motion_v / 2, 8, TRUE);
03166     plm_video_process_macroblock(self, s->cb.data, d->cb.data, motion_h / 2, motion_v / 2, 8, TRUE);
03167 }
03168
03169 #define PLM_BLOCK_SET(DEST, DEST_INDEX, DEST_WIDTH, SOURCE_INDEX, SOURCE_WIDTH, BLOCK_SIZE, OP) do { \
03170     int dest_scan = DEST_WIDTH - BLOCK_SIZE; \
03171     int source_scan = SOURCE_WIDTH - BLOCK_SIZE; \
03172     for (int y = 0; y < BLOCK_SIZE; y++) { \
03173         for (int x = 0; x < BLOCK_SIZE; x++) { \
03174             DEST[DEST_INDEX] = OP; \
03175             SOURCE_INDEX++; DEST_INDEX++; \
03176         } \
03177         SOURCE_INDEX += source_scan; \
03178         DEST_INDEX += dest_scan; \
03179     } } while(FALSE)
03180
03181 void plm_video_process_macroblock(
03182     plm_video_t *self, uint8_t *s, uint8_t *d,
03183     int motion_h, int motion_v, int block_size, int interpolate
03184 ) {
03185     int dw = self->mb_width * block_size;
03186
03187     int hp = motion_h >> 1;
03188     int vp = motion_v >> 1;
03189     int odd_h = (motion_h & 1) == 1;
03190     int odd_v = (motion_v & 1) == 1;
03191
03192     unsigned int si = ((self->mb_row * block_size) + vp) * dw + (self->mb_col * block_size) + hp;
03193     unsigned int di = (self->mb_row * dw + self->mb_col) * block_size;
03194
03195     unsigned int max_address = (dw * (self->mb_height * block_size - block_size + 1) - block_size);
03196     if (si > max_address || di > max_address) {
03197         return; // corrupt video
03198     }
03199
03200     #define PLM_MB_CASE(INTERPOLATE, ODD_H, ODD_V, OP) \
03201         case ((INTERPOLATE << 2) | (ODD_H << 1) | (ODD_V)): \
03202             PLM_BLOCK_SET(d, di, dw, si, dw, block_size, OP); \
03203             break
03204
03205     switch ((interpolate << 2) | (odd_h << 1) | (odd_v)) {
03206         PLM_MB_CASE(0, 0, 0, (s[si]));
03207         PLM_MB_CASE(0, 0, 1, (s[si] + s[si + dw] + 1) >> 1);
03208         PLM_MB_CASE(0, 1, 0, (s[si] + s[si + 1] + 1) >> 1);
03209         PLM_MB_CASE(0, 1, 1, (s[si] + s[si + 1] + s[si + dw] + s[si + dw + 1] + 2) >> 2);
03210
03211         PLM_MB_CASE(1, 0, 0, (d[di] + (s[si]) + 1) >> 1);
03212         PLM_MB_CASE(1, 0, 1, (d[di] + ((s[si] + s[si + dw] + 1) >> 1) + 1) >> 1);
03213         PLM_MB_CASE(1, 1, 0, (d[di] + ((s[si] + s[si + 1] + 1) >> 1) + 1) >> 1);
03214         PLM_MB_CASE(1, 1, 1, (d[di] + ((s[si] + s[si + 1] + s[si + dw] + s[si + dw + 1] + 2) >> 2) + 1)
03215     >> 1);
03216     }
03217     #undef PLM_MB_CASE
03218 }
03219
03220 void plm_video_decode_block(plm_video_t *self, int block) {
03221     int n = 0;

```

```

03223     uint8_t *quant_matrix;
03224
03225     // Decode DC coefficient of intra-coded blocks
03226     if (self->macroblock_intra) {
03227         int predictor;
03228         int dct_size;
03229
03230         // DC prediction
03231         int plane_index = block > 3 ? block - 3 : 0;
03232         predictor = self->dc_predictor[plane_index];
03233         dct_size = plm_buffer_read_vlc(self->buffer, PLM_VIDEO_DCT_SIZE[plane_index]);
03234
03235         // Read DC coeff
03236         if (dct_size > 0) {
03237             int differential = plm_buffer_read(self->buffer, dct_size);
03238             if ((differential & (1 « (dct_size - 1))) != 0) {
03239                 self->block_data[0] = predictor + differential;
03240             }
03241             else {
03242                 self->block_data[0] = predictor + (-(1 « dct_size) | (differential + 1));
03243             }
03244         }
03245         else {
03246             self->block_data[0] = predictor;
03247         }
03248
03249         // Save predictor value
03250         self->dc_predictor[plane_index] = self->block_data[0];
03251
03252         // Dequantize + premultiply
03253         self->block_data[0] «= (3 + 5);
03254
03255         quant_matrix = self->intra_quant_matrix;
03256         n = 1;
03257     }
03258     else {
03259         quant_matrix = self->non_intra_quant_matrix;
03260     }
03261
03262     // Decode AC coefficients (+DC for non-intra)
03263     int level = 0;
03264     while (TRUE) {
03265         int run = 0;
03266         uint16_t coeff = plm_buffer_read_vlc_uint(self->buffer, PLM_VIDEO_DCT_COEFF);
03267
03268         if ((coeff == 0x0001) && (n > 0) && (plm_buffer_read(self->buffer, 1) == 0)) {
03269             // end_of_block
03270             break;
03271         }
03272         if (coeff == 0xffff) {
03273             // escape
03274             run = plm_buffer_read(self->buffer, 6);
03275             level = plm_buffer_read(self->buffer, 8);
03276             if (level == 0) {
03277                 level = plm_buffer_read(self->buffer, 8);
03278             }
03279             else if (level == 128) {
03280                 level = plm_buffer_read(self->buffer, 8) - 256;
03281             }
03282             else if (level > 128) {
03283                 level = level - 256;
03284             }
03285         }
03286         else {
03287             run = coeff » 8;
03288             level = coeff & 0xff;
03289             if (plm_buffer_read(self->buffer, 1)) {
03290                 level = -level;
03291             }
03292         }
03293
03294         n += run;
03295         if (n < 0 || n >= 64) {
03296             return; // invalid
03297         }
03298
03299         int de_zig_zagged = PLM_VIDEO_ZIG_ZAG[n];
03300         n++;
03301
03302         // Dequantize, oddify, clip
03303         level «= 1;
03304         if (!self->macroblock_intra) {
03305             level += (level < 0 ? -1 : 1);
03306         }
03307         level = (level * self->quantizer_scale * quant_matrix[de_zig_zagged]) » 4;
03308         if ((level & 1) == 0) {
03309             level -= level > 0 ? 1 : -1;

```

```

03310         }
03311         if (level > 2047) {
03312             level = 2047;
03313         }
03314         else if (level < -2048) {
03315             level = -2048;
03316         }
03317
03318         // Save premultiplied coefficient
03319         self->block_data[de_zig_zagged] = level * PLM_VIDEO_PREMULTIPLIER_MATRIX[de_zig_zagged];
03320     }
03321
03322     // Move block to its place
03323     uint8_t *d;
03324     int dw;
03325     int di;
03326
03327     if (block < 4) {
03328         d = self->frame_current.y.data;
03329         dw = self->luma_width;
03330         di = (self->mb_row * self->luma_width + self->mb_col) << 4;
03331         if ((block & 1) != 0) {
03332             di += 8;
03333         }
03334         if ((block & 2) != 0) {
03335             di += self->luma_width << 3;
03336         }
03337     }
03338     else {
03339         d = (block == 4) ? self->frame_current.cb.data : self->frame_current.cr.data;
03340         dw = self->chroma_width;
03341         di = ((self->mb_row * self->luma_width) << 2) + (self->mb_col << 3);
03342     }
03343
03344     int *s = self->block_data;
03345     int si = 0;
03346     if (self->macroblock_intra) {
03347         // Overwrite (no prediction)
03348         if (n == 1) {
03349             int clamped = plm_clamp((s[0] + 128) >> 8);
03350             PLM_BLOCK_SET(d, di, dw, si, 8, 8, clamped);
03351             s[0] = 0;
03352         }
03353         else {
03354             plm_video_idct(s);
03355             PLM_BLOCK_SET(d, di, dw, si, 8, 8, plm_clamp(s[si]));
03356             memset(self->block_data, 0, sizeof(self->block_data));
03357         }
03358     }
03359     else {
03360         // Add data to the predicted macroblock
03361         if (n == 1) {
03362             int value = (s[0] + 128) >> 8;
03363             PLM_BLOCK_SET(d, di, dw, si, 8, 8, plm_clamp(d[di] + value));
03364             s[0] = 0;
03365         }
03366         else {
03367             plm_video_idct(s);
03368             PLM_BLOCK_SET(d, di, dw, si, 8, 8, plm_clamp(d[di] + s[si]));
03369             memset(self->block_data, 0, sizeof(self->block_data));
03370         }
03371     }
03372 }
03373
03374 void plm_video_idct(int *block) {
03375     int
03376         b1, b3, b4, b6, b7, tmp1, tmp2, m0,
03377         x0, x1, x2, x3, x4, y3, y4, y5, y6, y7;
03378
03379     // Transform columns
03380     for (int i = 0; i < 8; ++i) {
03381         b1 = block[4 * 8 + i];
03382         b3 = block[2 * 8 + i] + block[6 * 8 + i];
03383         b4 = block[5 * 8 + i] - block[3 * 8 + i];
03384         tmp1 = block[1 * 8 + i] + block[7 * 8 + i];
03385         tmp2 = block[3 * 8 + i] + block[5 * 8 + i];
03386         b6 = block[1 * 8 + i] - block[7 * 8 + i];
03387         b7 = tmp1 + tmp2;
03388         m0 = block[0 * 8 + i];
03389         x4 = ((b6 * 473 - b4 * 196 + 128) >> 8) - b7;
03390         x0 = x4 - (((tmp1 - tmp2) * 362 + 128) >> 8);
03391         x1 = m0 - b1;
03392         x2 = (((block[2 * 8 + i] - block[6 * 8 + i]) * 362 + 128) >> 8) - b3;
03393         x3 = m0 + b1;
03394         y3 = x1 + x2;
03395         y4 = x3 + b3;
03396         y5 = x1 - x2;

```

```

03397     y6 = x3 - b3;
03398     y7 = -x0 - ((b4 * 473 + b6 * 196 + 128) >> 8);
03399     block[0 * 8 + i] = b7 + y4;
03400     block[1 * 8 + i] = x4 + y3;
03401     block[2 * 8 + i] = y5 - x0;
03402     block[3 * 8 + i] = y6 - y7;
03403     block[4 * 8 + i] = y6 + y7;
03404     block[5 * 8 + i] = x0 + y5;
03405     block[6 * 8 + i] = y3 - x4;
03406     block[7 * 8 + i] = y4 - b7;
03407 }
03408
03409 // Transform rows
03410 for (int i = 0; i < 64; i += 8) {
03411     b1 = block[4 + i];
03412     b3 = block[2 + i] + block[6 + i];
03413     b4 = block[5 + i] - block[3 + i];
03414     tmp1 = block[1 + i] + block[7 + i];
03415     tmp2 = block[3 + i] + block[5 + i];
03416     b6 = block[1 + i] - block[7 + i];
03417     b7 = tmp1 + tmp2;
03418     m0 = block[0 + i];
03419     x4 = ((b6 * 473 - b4 * 196 + 128) >> 8) - b7;
03420     x0 = x4 - (((tmp1 - tmp2) * 362 + 128) >> 8);
03421     x1 = m0 - b1;
03422     x2 = (((block[2 + i] - block[6 + i]) * 362 + 128) >> 8) - b3;
03423     x3 = m0 + b1;
03424     y3 = x1 + x2;
03425     y4 = x3 + b3;
03426     y5 = x1 - x2;
03427     y6 = x3 - b3;
03428     y7 = -x0 - ((b4 * 473 + b6 * 196 + 128) >> 8);
03429     block[0 + i] = (b7 + y4 + 128) >> 8;
03430     block[1 + i] = (x4 + y3 + 128) >> 8;
03431     block[2 + i] = (y5 - x0 + 128) >> 8;
03432     block[3 + i] = (y6 - y7 + 128) >> 8;
03433     block[4 + i] = (y6 + y7 + 128) >> 8;
03434     block[5 + i] = (x0 + y5 + 128) >> 8;
03435     block[6 + i] = (y3 - x4 + 128) >> 8;
03436     block[7 + i] = (y4 - b7 + 128) >> 8;
03437 }
03438 }
03439
03440 // YCbCr conversion following the BT.601 standard:
03441 // https://infogalactic.com/info/YCbCr#ITU-R_BT.601_conversion
03442
03443 #define PLM_PUT_PIXEL(RI, GI, BI, Y_OFFSET, DEST_OFFSET) \
03444     y = ((frame->y.data[y_index + Y_OFFSET]-16) * 76309) >> 16; \
03445     dest[d_index + DEST_OFFSET + RI] = plm_clamp(y + r); \
03446     dest[d_index + DEST_OFFSET + GI] = plm_clamp(y - g); \
03447     dest[d_index + DEST_OFFSET + BI] = plm_clamp(y + b);
03448
03449 #define PLM_DEFINE_FRAME_CONVERT_FUNCTION(NAME, BYTES_PER_PIXEL, RI, GI, BI) \
03450 void NAME(plm_frame_t *frame, uint8_t *dest, int stride) { \
03451     int cols = frame->width >> 1; \
03452     int rows = frame->height >> 1; \
03453     int yw = frame->y.width; \
03454     int cw = frame->cb.width; \
03455     for (int row = 0; row < rows; row++) { \
03456         int c_index = row * cw; \
03457         int y_index = row * 2 * yw; \
03458         int d_index = row * 2 * stride; \
03459         for (int col = 0; col < cols; col++) { \
03460             int y; \
03461             int cr = frame->cr.data[c_index] - 128; \
03462             int cb = frame->cb.data[c_index] - 128; \
03463             int r = (cr * 104597) >> 16; \
03464             int g = (cb * 25674 + cr * 53278) >> 16; \
03465             int b = (cb * 132201) >> 16; \
03466             PLM_PUT_PIXEL(RI, GI, BI, 0, 0); \
03467             PLM_PUT_PIXEL(RI, GI, BI, 1, BYTES_PER_PIXEL); \
03468             PLM_PUT_PIXEL(RI, GI, BI, yw, stride); \
03469             PLM_PUT_PIXEL(RI, GI, BI, yw + 1, stride + BYTES_PER_PIXEL); \
03470             c_index += 1; \
03471             y_index += 2; \
03472             d_index += 2 * BYTES_PER_PIXEL; \
03473         } \
03474     } \
03475 }
03476
03477 PLM_DEFINE_FRAME_CONVERT_FUNCTION(plm_frame_to_rgb, 3, 0, 1, 2)
03478 PLM_DEFINE_FRAME_CONVERT_FUNCTION(plm_frame_to_bgr, 3, 2, 1, 0)
03479 PLM_DEFINE_FRAME_CONVERT_FUNCTION(plm_frame_to_rgba, 4, 0, 1, 2)
03480 PLM_DEFINE_FRAME_CONVERT_FUNCTION(plm_frame_to_bgra, 4, 2, 1, 0)
03481 PLM_DEFINE_FRAME_CONVERT_FUNCTION(plm_frame_to_argb, 4, 1, 2, 3)
03482 PLM_DEFINE_FRAME_CONVERT_FUNCTION(plm_frame_to_abgr, 4, 3, 2, 1)
03483

```

```

03484
03485 #undef PLM_PUT_PIXEL
03486 #undef PLM_DEFINE_FRAME_CONVERT_FUNCTION
03487
03488
03489
03490 // -----
03491 // plm_audio implementation
03492
03493 // Based on kjmp2 by Martin J. Fiedler
03494 // http://keyj.emphy.de/kjmp2/
03495
03496 static const int PLM_AUDIO_FRAME_SYNC = 0x7ff;
03497
03498 static const int PLM_AUDIO_MPEG_2_5 = 0x0;
03499 static const int PLM_AUDIO_MPEG_2 = 0x2;
03500 static const int PLM_AUDIO_MPEG_1 = 0x3;
03501
03502 static const int PLM_AUDIO_LAYER_III = 0x1;
03503 static const int PLM_AUDIO_LAYER_II = 0x2;
03504 static const int PLM_AUDIO_LAYER_I = 0x3;
03505
03506 static const int PLM_AUDIO_MODE_STEREO = 0x0;
03507 static const int PLM_AUDIO_MODE_JOINT_STEREO = 0x1;
03508 static const int PLM_AUDIO_MODE_DUAL_CHANNEL = 0x2;
03509 static const int PLM_AUDIO_MODE_MONO = 0x3;
03510
03511 static const unsigned short PLM_AUDIO_SAMPLE_RATE[] = {
03512     44100, 48000, 32000, 0, // MPEG-1
03513     22050, 24000, 16000, 0 // MPEG-2
03514 };
03515
03516 static const short PLM_AUDIO_BIT_RATE[] = {
03517     32, 48, 56, 64, 80, 96, 112, 128, 160, 192, 224, 256, 320, 384, // MPEG-1
03518     8, 16, 24, 32, 40, 48, 56, 64, 80, 96, 112, 128, 144, 160 // MPEG-2
03519 };
03520
03521 static const int PLM_AUDIO_SCALEFACTOR_BASE[] = {
03522     0x02000000, 0x01965FEA, 0x01428A30
03523 };
03524
03525 static const float PLM_AUDIO_SYNTHESIS_WINDOW[] = {
03526     0.0, -0.5, -0.5, -0.5, -0.5, -0.5,
03527     -0.5, -1.0, -1.0, -1.0, -1.0, -1.5,
03528     -1.5, -2.0, -2.0, -2.5, -2.5, -3.0,
03529     -3.5, -3.5, -4.0, -4.5, -5.0, -5.5,
03530     -6.5, -7.0, -8.0, -8.5, -9.5, -10.5,
03531     -12.0, -13.0, -14.5, -15.5, -17.5, -19.0,
03532     -20.5, -22.5, -24.5, -26.5, -29.0, -31.5,
03533     -34.0, -36.5, -39.5, -42.5, -45.5, -48.5,
03534     -52.0, -55.5, -58.5, -62.5, -66.0, -69.5,
03535     -73.5, -77.0, -80.5, -84.5, -88.0, -91.5,
03536     -95.0, -98.0, -101.0, -104.0, -106.5, -109.0,
03537     111.0, 112.5, 113.5, 114.0, 114.0, 113.5,
03538     112.0, 110.5, 107.5, 104.0, 100.0, 94.5,
03539     88.5, 81.5, 73.0, 63.5, 53.0, 41.5,
03540     28.5, 14.5, -1.0, -18.0, -36.0, -55.5,
03541     -76.5, -98.5, -122.0, -147.0, -173.5, -200.5,
03542     -229.5, -259.5, -290.5, -322.5, -355.5, -389.5,
03543     -424.0, -459.5, -495.5, -532.0, -568.5, -605.0,
03544     -641.5, -678.0, -714.0, -749.0, -783.5, -817.0,
03545     -849.0, -879.5, -908.5, -935.0, -959.5, -981.0,
03546     -1000.5, -1016.0, -1028.5, -1037.5, -1042.5, -1043.5,
03547     -1040.0, -1031.5, 1018.5, 1000.0, 976.0, 946.5,
03548     911.0, 869.5, 822.0, 767.5, 707.0, 640.0,
03549     565.5, 485.0, 397.0, 302.5, 201.0, 92.5,
03550     -22.5, -144.0, -272.5, -407.0, -547.5, -694.0,
03551     -846.0, -1003.0, -1165.0, -1331.5, -1502.0, -1675.5,
03552     -1852.5, -2031.5, -2212.5, -2394.0, -2576.5, -2758.5,
03553     -2939.5, -3118.5, -3294.5, -3467.5, -3635.5, -3798.5,
03554     -3955.0, -4104.5, -4245.5, -4377.5, -4499.0, -4609.5,
03555     -4708.0, -4792.5, -4863.5, -4919.0, -4958.0, -4979.5,
03556     -4983.0, -4967.5, -4931.5, -4875.0, -4796.0, -4694.5,
03557     -4569.5, -4420.0, -4246.0, -4046.0, -3820.0, -3567.0,
03558     3287.0, 2979.5, 2644.0, 2280.5, 1888.0, 1467.5,
03559     1018.5, 541.0, 35.0, -499.0, -1061.0, -1650.0,
03560     -2266.5, -2909.0, -3577.0, -4270.0, -4987.5, -5727.5,
03561     -6490.0, -7274.0, -8077.5, -8899.5, -9739.0, -10594.5,
03562     -11464.5, -12347.0, -13241.0, -14144.5, -15056.0, -15973.5,
03563     -16895.5, -17820.0, -18744.5, -19668.0, -20588.0, -21503.0,
03564     -22410.5, -23308.5, -24195.0, -25068.5, -25926.5, -26767.0,
03565     -27589.0, -28389.0, -29166.5, -29919.0, -30644.5, -31342.0,
03566     -32009.5, -32645.0, -33247.0, -33814.5, -34346.0, -34839.5,
03567     -35295.0, -35710.0, -36084.5, -36417.5, -36707.5, -36954.0,
03568     -37156.5, -37315.0, -37428.0, -37496.0, -37519.0, -37496.0,
03569     37428.0, 37315.0, 37156.5, 36954.0, 36707.5, 36417.5,
03570     36084.5, 35710.0, 35295.0, 34839.5, 34346.0, 33814.5,

```

```

03571     33247.0,   32645.0,   32009.5,   31342.0,   30644.5,   29919.0,
03572     29166.5,   28389.0,   27589.0,   26767.0,   25926.5,   25068.5,
03573     24195.0,   23308.5,   22410.5,   21503.0,   20588.0,   19668.0,
03574     18744.5,   17820.0,   16895.5,   15973.5,   15056.0,   14144.5,
03575     13241.0,   12347.0,   11464.5,   10594.5,   9739.0,    8899.5,
03576     8077.5,    7274.0,    6490.0,    5727.5,    4987.5,    4270.0,
03577     3577.0,    2909.0,    2266.5,    1650.0,    1061.0,    499.0,
03578     -35.0,     -541.0,   -1018.5,   -1467.5,   -1888.0,   -2280.5,
03579     -2644.0,   -2979.5,   3287.0,    3567.0,    3820.0,    4046.0,
03580     4246.0,    4420.0,    4569.5,    4694.5,    4796.0,    4875.0,
03581     4931.5,    4967.5,    4983.0,    4979.5,    4958.0,    4919.0,
03582     4863.5,    4792.5,    4708.0,    4609.5,    4499.0,    4377.5,
03583     4245.5,    4104.5,    3955.0,    3798.5,    3635.5,    3467.5,
03584     3294.5,    3118.5,    2939.5,    2758.5,    2576.5,    2394.0,
03585     2212.5,    2031.5,    1852.5,    1675.5,    1502.0,    1331.5,
03586     1165.0,    1003.0,    846.0,     694.0,     547.5,     407.0,
03587     272.5,     144.0,     22.5,     -92.5,    -201.0,    -302.5,
03588     -397.0,    -485.0,    -565.5,    -640.0,    -707.0,    -767.5,
03589     -822.0,    -869.5,    -911.0,    -946.5,    -976.0,   -1000.0,
03590     1018.5,    1031.5,   1040.0,   1043.5,   1042.5,   1037.5,
03591     1028.5,    1016.0,   1000.5,   981.0,    959.5,    935.0,
03592     908.5,     879.5,    849.0,    817.0,    783.5,    749.0,
03593     714.0,     678.0,    641.5,    605.0,    568.5,    532.0,
03594     495.5,     459.5,    424.0,    389.5,    355.5,    322.5,
03595     290.5,     259.5,    229.5,    200.5,    173.5,    147.0,
03596     122.0,     98.5,     76.5,     55.5,     36.0,     18.0,
03597     1.0,      -14.5,    -28.5,    -41.5,    -53.0,    -63.5,
03598     -73.0,    -81.5,    -88.5,    -94.5,    -100.0,   -104.0,
03599     -107.5,   -110.5,   -112.0,   -113.5,   -114.0,   -114.0,
03600     -113.5,   -112.5,   -111.0,   -109.0,   106.5,    104.0,
03601     101.0,     98.0,     95.0,     91.5,     88.0,     84.5,
03602     80.5,      77.0,     73.5,     69.5,     66.0,     62.5,
03603     58.5,      55.5,     52.0,     48.5,     45.5,     42.5,
03604     39.5,      36.5,     34.0,     31.5,     29.0,     26.5,
03605     24.5,      22.5,     20.5,     19.0,     17.5,     15.5,
03606     14.5,      13.0,     12.0,     10.5,     9.5,      8.5,
03607     8.0,       7.0,      6.5,      5.5,      5.0,      4.5,
03608     4.0,       3.5,      3.5,      3.0,      2.5,      2.5,
03609     2.0,       2.0,      1.5,      1.5,      1.0,      1.0,
03610     1.0,       1.0,      0.5,      0.5,      0.5,      0.5,
03611     0.5,       0.5
03612 };
03613
03614 // Quantizer lookup, step 1: bitrate classes
03615 static const uint8_t PLM_AUDIO_QUANT_LUT_STEP_1[2][16] = {
03616     // 32, 48, 56, 64, 80, 96, 112, 128, 160, 192, 224, 256, 320, 384 <- bitrate
03617     { 0, 0, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2 }, // mono
03618     // 16, 24, 28, 32, 40, 48, 56, 64, 80, 96, 112, 128, 160, 192 <- bitrate / chan
03619     { 0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 2 } // stereo
03620 };
03621
03622 // Quantizer lookup, step 2: bitrate class, sample rate -> B2 table idx, sblimit
03623 #define PLM_AUDIO_QUANT_TAB_A (27 | 64) // Table 3-B.2a: high-rate, sblimit = 27
03624 #define PLM_AUDIO_QUANT_TAB_B (30 | 64) // Table 3-B.2b: high-rate, sblimit = 30
03625 #define PLM_AUDIO_QUANT_TAB_C 8 // Table 3-B.2c: low-rate, sblimit = 8
03626 #define PLM_AUDIO_QUANT_TAB_D 12 // Table 3-B.2d: low-rate, sblimit = 12
03627
03628 static const uint8_t QUANT_LUT_STEP_2[3][3] = {
03629     //44.1 kHz,          48 kHz,          32 kHz
03630     { PLM_AUDIO_QUANT_TAB_C, PLM_AUDIO_QUANT_TAB_C, PLM_AUDIO_QUANT_TAB_D }, // 32 - 48 kbit/sec/ch
03631     { PLM_AUDIO_QUANT_TAB_A, PLM_AUDIO_QUANT_TAB_A, PLM_AUDIO_QUANT_TAB_A }, // 56 - 80 kbit/sec/ch
03632     { PLM_AUDIO_QUANT_TAB_B, PLM_AUDIO_QUANT_TAB_A, PLM_AUDIO_QUANT_TAB_B } // 96+ kbit/sec/ch
03633 };
03634
03635 // Quantizer lookup, step 3: B2 table, subband -> nbal, row index
03636 // (upper 4 bits: nbal, lower 4 bits: row index)
03637 static const uint8_t PLM_AUDIO_QUANT_LUT_STEP_3[3][32] = {
03638     // Low-rate table (3-B.2c and 3-B.2d)
03639     {
03640         0x44, 0x44,
03641         0x34, 0x34, 0x34, 0x34, 0x34, 0x34, 0x34, 0x34, 0x34
03642     },
03643     // High-rate table (3-B.2a and 3-B.2b)
03644     {
03645         0x43, 0x43, 0x43,
03646         0x42, 0x42, 0x42, 0x42, 0x42, 0x42, 0x42, 0x42,
03647         0x31, 0x31, 0x31, 0x31, 0x31, 0x31, 0x31, 0x31, 0x31, 0x31,
03648         0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20
03649     },
03650     // MPEG-2 LSR table (B.2 in ISO 13818-3)
03651     {
03652         0x45, 0x45, 0x45, 0x45,
03653         0x34, 0x34, 0x34, 0x34, 0x34, 0x34, 0x34,
03654         0x24, 0x24, 0x24, 0x24, 0x24, 0x24, 0x24, 0x24,
03655         0x24, 0x24, 0x24, 0x24, 0x24, 0x24, 0x24, 0x24
03656     }
03657 };

```

```

03658
03659 // Quantizer lookup, step 4: table row, allocation[] value -> quant table index
03660 static const uint8_t PLM_AUDIO_QUANT_LUT_STEP_4[6][16] = {
03661 { 0, 1, 2, 17 },
03662 { 0, 1, 2, 3, 4, 5, 6, 17 },
03663 { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 17 },
03664 { 0, 1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17 },
03665 { 0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17 },
03666 { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 }
03667 };
03668
03669 typedef struct plm_quantizer_spec_t {
03670     unsigned short levels;
03671     unsigned char group;
03672     unsigned char bits;
03673 } plm_quantizer_spec_t;
03674
03675 static const plm_quantizer_spec_t PLM_AUDIO_QUANT_TAB[] = {
03676 { 3, 1, 5 }, // 1
03677 { 5, 1, 7 }, // 2
03678 { 7, 0, 3 }, // 3
03679 { 9, 1, 10 }, // 4
03680 { 15, 0, 4 }, // 5
03681 { 31, 0, 5 }, // 6
03682 { 63, 0, 6 }, // 7
03683 { 127, 0, 7 }, // 8
03684 { 255, 0, 8 }, // 9
03685 { 511, 0, 9 }, // 10
03686 { 1023, 0, 10 }, // 11
03687 { 2047, 0, 11 }, // 12
03688 { 4095, 0, 12 }, // 13
03689 { 8191, 0, 13 }, // 14
03690 { 16383, 0, 14 }, // 15
03691 { 32767, 0, 15 }, // 16
03692 { 65535, 0, 16 } // 17
03693 };
03694
03695 struct plm_audio_t {
03696     double time;
03697     int samples_decoded;
03698     int samplerate_index;
03699     int bitrate_index;
03700     int version;
03701     int layer;
03702     int mode;
03703     int bound;
03704     int v_pos;
03705     int next_frame_data_size;
03706     int has_header;
03707
03708     plm_buffer_t *buffer;
03709     int destroy_buffer_when_done;
03710
03711     const plm_quantizer_spec_t *allocation[2][32];
03712     uint8_t scale_factor_info[2][32];
03713     int scale_factor[2][32][3];
03714     int sample[2][32][3];
03715
03716     plm_samples_t samples;
03717     float D[1024];
03718     float V[2][1024];
03719     float U[32];
03720 };
03721
03722 int plm_audio_find_frame_sync(plm_audio_t *self);
03723 int plm_audio_decode_header(plm_audio_t *self);
03724 void plm_audio_decode_frame(plm_audio_t *self);
03725 const plm_quantizer_spec_t *plm_audio_read_allocation(plm_audio_t *self, int sb, int tab3);
03726 void plm_audio_read_samples(plm_audio_t *self, int ch, int sb, int part);
03727 void plm_audio_idct36(int s[32][3], int ss, float *d, int dp);
03728
03729 plm_audio_t *plm_audio_create_with_buffer(plm_buffer_t *buffer, int destroy_when_done) {
03730     plm_audio_t *self = (plm_audio_t *)PLM_MALLOC(sizeof(plm_audio_t));
03731     memset(self, 0, sizeof(plm_audio_t));
03732
03733     self->samples.count = PLM_AUDIO_SAMPLES_PER_FRAME;
03734     self->buffer = buffer;
03735     self->destroy_buffer_when_done = destroy_when_done;
03736     self->samplerate_index = 3; // Indicates 0
03737
03738     memcpy(self->D, PLM_AUDIO_SYNTHESIS_WINDOW, 512 * sizeof(float));
03739     memcpy(self->D + 512, PLM_AUDIO_SYNTHESIS_WINDOW, 512 * sizeof(float));
03740
03741     // Attempt to decode first header
03742     self->next_frame_data_size = plm_audio_decode_header(self);
03743
03744     return self;

```

```

03745 }
03746
03747 void plm_audio_destroy(plm_audio_t *self) {
03748     if (self->destroy_buffer_when_done) {
03749         plm_buffer_destroy(self->buffer);
03750     }
03751     PLM_FREE(self);
03752 }
03753
03754 int plm_audio_has_header(plm_audio_t *self) {
03755     if (self->has_header) {
03756         return TRUE;
03757     }
03758
03759     self->next_frame_data_size = plm_audio_decode_header(self);
03760     return self->has_header;
03761 }
03762
03763 int plm_audio_get_samplerate(plm_audio_t *self) {
03764     return plm_audio_has_header(self)
03765         ? PLM_AUDIO_SAMPLE_RATE[self->samplerate_index]
03766         : 0;
03767 }
03768
03769 double plm_audio_get_time(plm_audio_t *self) {
03770     return self->time;
03771 }
03772
03773 void plm_audio_set_time(plm_audio_t *self, double time) {
03774     self->samples_decoded = time *
03775         (double)PLM_AUDIO_SAMPLE_RATE[self->samplerate_index];
03776     self->time = time;
03777 }
03778
03779 void plm_audio_rewind(plm_audio_t *self) {
03780     plm_buffer_rewind(self->buffer);
03781     self->time = 0;
03782     self->samples_decoded = 0;
03783     self->next_frame_data_size = 0;
03784 }
03785
03786 int plm_audio_has-ended(plm_audio_t *self) {
03787     return plm_buffer_has-ended(self->buffer);
03788 }
03789
03790 plm_samples_t *plm_audio_decode(plm_audio_t *self) {
03791     // Do we have at least enough information to decode the frame header?
03792     if (!self->next_frame_data_size) {
03793         if (!plm_buffer_has(self->buffer, 48)) {
03794             return NULL;
03795         }
03796         self->next_frame_data_size = plm_audio_decode_header(self);
03797     }
03798
03799     if (
03800         self->next_frame_data_size == 0 ||
03801         !plm_buffer_has(self->buffer, self->next_frame_data_size << 3)
03802     ) {
03803         return NULL;
03804     }
03805
03806     plm_audio_decode_frame(self);
03807     self->next_frame_data_size = 0;
03808
03809     self->samples.time = self->time;
03810
03811     self->samples_decoded += PLM_AUDIO_SAMPLES_PER_FRAME;
03812     self->time = (double)self->samples_decoded /
03813         (double)PLM_AUDIO_SAMPLE_RATE[self->samplerate_index];
03814
03815     return &self->samples;
03816 }
03817
03818 int plm_audio_find_frame_sync(plm_audio_t *self) {
03819     size_t i;
03820     for (i = self->buffer->bit_index >> 3; i < self->buffer->length-1; i++) {
03821         if (
03822             self->buffer->bytes[i] == 0xFF &&
03823             (self->buffer->bytes[i+1] & 0xFE) == 0xFC
03824         ) {
03825             self->buffer->bit_index = ((i+1) << 3) + 3;
03826             return TRUE;
03827         }
03828     }
03829     self->buffer->bit_index = (i + 1) << 3;
03830     return FALSE;
03831 }

```

```

03832
03833 int plm_audio_decode_header(plm_audio_t *self) {
03834     if (!plm_buffer_has(self->buffer, 48)) {
03835         return 0;
03836     }
03837
03838     plm_buffer_skip_bytes(self->buffer, 0x00);
03839     int sync = plm_buffer_read(self->buffer, 11);
03840
03841
03842     // Attempt to resync if no syncword was found. This sucks balls. The MP2
03843     // stream contains a syncword just before every frame (11 bits set to 1).
03844     // However, this syncword is not guaranteed to not occur elsewhere in the
03845     // stream. So, if we have to resync, we also have to check if the header
03846     // (samplerate, bitrate) differs from the one we had before. This all
03847     // may still lead to garbage data being decoded :/
03848
03849     if (sync != PLM_AUDIO_FRAME_SYNC && !plm_audio_find_frame_sync(self)) {
03850         return 0;
03851     }
03852
03853     self->version = plm_buffer_read(self->buffer, 2);
03854     self->layer = plm_buffer_read(self->buffer, 2);
03855     int hasCRC = !plm_buffer_read(self->buffer, 1);
03856
03857     if (
03858         self->version != PLM_AUDIO_MPEG_1 ||
03859         self->layer != PLM_AUDIO_LAYER_II
03860     ) {
03861         return 0;
03862     }
03863
03864     int bitrate_index = plm_buffer_read(self->buffer, 4) - 1;
03865     if (bitrate_index > 13) {
03866         return 0;
03867     }
03868
03869     int samplerate_index = plm_buffer_read(self->buffer, 2);
03870     if (samplerate_index == 3) {
03871         return 0;
03872     }
03873
03874     int padding = plm_buffer_read(self->buffer, 1);
03875     plm_buffer_skip(self->buffer, 1); // f_private
03876     int mode = plm_buffer_read(self->buffer, 2);
03877
03878     // If we already have a header, make sure the samplerate, bitrate and mode
03879     // are still the same, otherwise we might have missed sync.
03880     if (
03881         self->has_header && (
03882             self->bitrate_index != bitrate_index ||
03883             self->samplerate_index != samplerate_index ||
03884             self->mode != mode
03885         )
03886     ) {
03887         return 0;
03888     }
03889
03890     self->bitrate_index = bitrate_index;
03891     self->samplerate_index = samplerate_index;
03892     self->mode = mode;
03893     self->has_header = TRUE;
03894
03895     // Parse the mode_extension, set up the stereo bound
03896     if (mode == PLM_AUDIO_MODE_JOINT_STEREO) {
03897         self->bound = (plm_buffer_read(self->buffer, 2) + 1) « 2;
03898     }
03899     else {
03900         plm_buffer_skip(self->buffer, 2);
03901         self->bound = (mode == PLM_AUDIO_MODE_MONO) ? 0 : 32;
03902     }
03903
03904     // Discard the last 4 bits of the header and the CRC value, if present
03905     plm_buffer_skip(self->buffer, 4); // copyright(1), original(1), emphasis(2)
03906     if (hasCRC) {
03907         plm_buffer_skip(self->buffer, 16);
03908     }
03909
03910     // Compute frame size, check if we have enough data to decode the whole
03911     // frame.
03912     int bitrate = PLM_AUDIO_BIT_RATE[self->bitrate_index];
03913     int samplerate = PLM_AUDIO_SAMPLE_RATE[self->samplerate_index];
03914     int frame_size = (144000 * bitrate / samplerate) + padding;
03915     return frame_size - (hasCRC ? 6 : 4);
03916 }
03917
03918 void plm_audio_decode_frame(plm_audio_t *self) {

```

```

03919 // Prepare the quantizer table lookups
03920 int tab3 = 0;
03921 int sblimit = 0;
03922
03923 int tab1 = (self->mode == PLM_AUDIO_MODE_MONO) ? 0 : 1;
03924 int tab2 = PLM_AUDIO_QUANT_LUT_STEP_1[tab1][self->bitrate_index];
03925 tab3 = QUANT_LUT_STEP_2[tab2][self->samplerate_index];
03926 sblimit = tab3 & 63;
03927 tab3 >>= 6;
03928
03929 if (self->bound > sblimit) {
03930     self->bound = sblimit;
03931 }
03932
03933 // Read the allocation information
03934 for (int sb = 0; sb < self->bound; sb++) {
03935     self->allocation[0][sb] = plm_audio_read_allocation(self, sb, tab3);
03936     self->allocation[1][sb] = plm_audio_read_allocation(self, sb, tab3);
03937 }
03938
03939 for (int sb = self->bound; sb < sblimit; sb++) {
03940     self->allocation[0][sb] =
03941         self->allocation[1][sb] =
03942             plm_audio_read_allocation(self, sb, tab3);
03943 }
03944
03945 // Read scale factor selector information
03946 int channels = (self->mode == PLM_AUDIO_MODE_MONO) ? 1 : 2;
03947 for (int sb = 0; sb < sblimit; sb++) {
03948     for (int ch = 0; ch < channels; ch++) {
03949         if (self->allocation[ch][sb]) {
03950             self->scale_factor_info[ch][sb] = plm_buffer_read(self->buffer, 2);
03951         }
03952     }
03953     if (self->mode == PLM_AUDIO_MODE_MONO) {
03954         self->scale_factor_info[1][sb] = self->scale_factor_info[0][sb];
03955     }
03956 }
03957
03958 // Read scale factors
03959 for (int sb = 0; sb < sblimit; sb++) {
03960     for (int ch = 0; ch < channels; ch++) {
03961         if (self->allocation[ch][sb]) {
03962             int *sf = self->scale_factor[ch][sb];
03963             switch (self->scale_factor_info[ch][sb]) {
03964                 case 0:
03965                     sf[0] = plm_buffer_read(self->buffer, 6);
03966                     sf[1] = plm_buffer_read(self->buffer, 6);
03967                     sf[2] = plm_buffer_read(self->buffer, 6);
03968                     break;
03969                 case 1:
03970                     sf[0] =
03971                         plm_buffer_read(self->buffer, 6);
03972                     sf[1] =
03973                         plm_buffer_read(self->buffer, 6);
03974                     sf[2] =
03975                         plm_buffer_read(self->buffer, 6);
03976                     break;
03977                 case 2:
03978                     sf[0] =
03979                         sf[1] =
03980                             sf[2] = plm_buffer_read(self->buffer, 6);
03981                     break;
03982                 case 3:
03983                     sf[0] = plm_buffer_read(self->buffer, 6);
03984                     sf[1] =
03985                         sf[2] = plm_buffer_read(self->buffer, 6);
03986                     break;
03987             }
03988         }
03989     }
03990     if (self->mode == PLM_AUDIO_MODE_MONO) {
03991         self->scale_factor[1][sb][0] = self->scale_factor[0][sb][0];
03992         self->scale_factor[1][sb][1] = self->scale_factor[0][sb][1];
03993         self->scale_factor[1][sb][2] = self->scale_factor[0][sb][2];
03994     }
03995 }
03996
03997 // Coefficient input and reconstruction
03998 int out_pos = 0;
03999 for (int part = 0; part < 3; part++) {
04000     for (int granule = 0; granule < 4; granule++) {
04001
04002         // Read the samples
04003         for (int sb = 0; sb < self->bound; sb++) {
04004             plm_audio_read_samples(self, 0, sb, part);
04005             plm_audio_read_samples(self, 1, sb, part);
04006         }
04007         for (int sb = self->bound; sb < sblimit; sb++) {
04008             plm_audio_read_samples(self, 0, sb, part);
04009         }
04010     }
04011 }
04012 }
```

```

04006         self->sample[1][sb][0] = self->sample[0][sb][0];
04007         self->sample[1][sb][1] = self->sample[0][sb][1];
04008         self->sample[1][sb][2] = self->sample[0][sb][2];
04009     }
0410     for (int sb = sblimit; sb < 32; sb++) {
0411         self->sample[0][sb][0] = 0;
0412         self->sample[0][sb][1] = 0;
0413         self->sample[0][sb][2] = 0;
0414         self->sample[1][sb][0] = 0;
0415         self->sample[1][sb][1] = 0;
0416         self->sample[1][sb][2] = 0;
0417     }
0418
0419     // Synthesis loop
0420     for (int p = 0; p < 3; p++) {
0421         // Shifting step
0422         self->v_pos = (self->v_pos - 64) & 1023;
0423
0424         for (int ch = 0; ch < 2; ch++) {
0425             plm_audio_idct36(self->sample[ch], p, self->V[ch], self->v_pos);
0426
0427             // Build U, windowing, calculate output
0428             memset(self->U, 0, sizeof(self->U));
0429
0430             int d_index = 512 - (self->v_pos >> 1);
0431             int v_index = (self->v_pos % 128) >> 1;
0432             while (v_index < 1024) {
0433                 for (int i = 0; i < 32; ++i) {
0434                     self->U[i] += self->D[d_index++] * self->V[ch][v_index++];
0435                 }
0436
0437                 v_index += 128 - 32;
0438                 d_index += 64 - 32;
0439             }
0440
0441             d_index -= (512 - 32);
0442             v_index = (128 - 32 + 1024) - v_index;
0443             while (v_index < 1024) {
0444                 for (int i = 0; i < 32; ++i) {
0445                     self->U[i] += self->D[d_index++] * self->V[ch][v_index++];
0446                 }
0447
0448                 v_index += 128 - 32;
0449                 d_index += 64 - 32;
0450             }
0451
0452             // Output samples
0453             #ifdef PLM_AUDIO_SEPARATE_CHANNELS
0454                 float *out_channel = ch == 0
0455                         ? self->samples.left
0456                         : self->samples.right;
0457                 for (int j = 0; j < 32; j++) {
0458                     out_channel[out_pos + j] = self->U[j] / 2147418112.0f;
0459                 }
0460             #else
0461                 for (int j = 0; j < 32; j++) {
0462                     self->samples.interleaved[((out_pos + j) << 1) + ch] =
0463                         self->U[j] / 2147418112.0f;
0464                 }
0465             #endif
0466         } // End of synthesis channel loop
0467         out_pos += 32;
0468     } // End of synthesis sub-block loop
0469
0470     } // Decoding of the granule finished
0471 }
0472
0473     plm_buffer_align(self->buffer);
0474 }
0475
0476 const plm_quantizer_spec_t *plm_audio_read_allocation(plm_audio_t *self, int sb, int tab3) {
0477     int tab4 = PLM_AUDIO_QUANT_LUT_STEP_3[tab3][sb];
0478     int qtab = PLM_AUDIO_QUANT_LUT_STEP_4[tab4 & 15][plm_buffer_read(self->buffer, tab4 >> 4)];
0479     return qtab ? (&PLM_AUDIO_QUANT_TAB[qtab - 1]) : 0;
0480 }
0481
0482 void plm_audio_read_samples(plm_audio_t *self, int ch, int sb, int part) {
0483     const plm_quantizer_spec_t *q = self->allocation[ch][sb];
0484     int sf = self->scale_factor[ch][sb][part];
0485     int *sample = self->sample[ch][sb];
0486     int val = 0;
0487
0488     if (!q) {
0489         // No bits allocated for this subband
0490         sample[0] = sample[1] = sample[2] = 0;
0491         return;
0492     }

```

```

04093
04094 // Resolve scalefactor
04095 if (sf == 63) {
04096     sf = 0;
04097 }
04098 else {
04099     int shift = (sf / 3) | 0;
04100     sf = (PLM_AUDIO_SCALEFACTOR_BASE[sf % 3] + ((1 << shift) >> 1)) >> shift;
04101 }
04102
04103 // Decode samples
04104 int adj = q->levels;
04105 if (q->group) {
04106     // Decode grouped samples
04107     val = plm_buffer_read(self->buffer, q->bits);
04108     sample[0] = val % adj;
04109     val /= adj;
04110     sample[1] = val % adj;
04111     sample[2] = val / adj;
04112 }
04113 else {
04114     // Decode direct samples
04115     sample[0] = plm_buffer_read(self->buffer, q->bits);
04116     sample[1] = plm_buffer_read(self->buffer, q->bits);
04117     sample[2] = plm_buffer_read(self->buffer, q->bits);
04118 }
04119
04120 // Postmultiply samples
04121 int scale = 65536 / (adj + 1);
04122 adj = ((adj + 1) >> 1) - 1;
04123
04124 val = (adj - sample[0]) * scale;
04125 sample[0] = (val * (sf >> 12) + ((val * (sf & 4095) + 2048) >> 12)) >> 12;
04126
04127 val = (adj - sample[1]) * scale;
04128 sample[1] = (val * (sf >> 12) + ((val * (sf & 4095) + 2048) >> 12)) >> 12;
04129
04130 val = (adj - sample[2]) * scale;
04131 sample[2] = (val * (sf >> 12) + ((val * (sf & 4095) + 2048) >> 12)) >> 12;
04132 }
04133
04134 void plm_audio_idct36(int s[32][3], int ss, float *d, int dp) {
04135     float t01, t02, t03, t04, t05, t06, t07, t08, t09, t10, t11, t12,
04136         t13, t14, t15, t16, t17, t18, t19, t20, t21, t22, t23, t24,
04137         t25, t26, t27, t28, t29, t30, t31, t32, t33;
04138
04139     t01 = (float)(s[0][ss] + s[31][ss]); t02 = (float)(s[0][ss] - s[31][ss]) * 0.500602998235f;
04140     t03 = (float)(s[1][ss] + s[30][ss]); t04 = (float)(s[1][ss] - s[30][ss]) * 0.505470959898f;
04141     t05 = (float)(s[2][ss] + s[29][ss]); t06 = (float)(s[2][ss] - s[29][ss]) * 0.515447309923f;
04142     t07 = (float)(s[3][ss] + s[28][ss]); t08 = (float)(s[3][ss] - s[28][ss]) * 0.53104259109f;
04143     t09 = (float)(s[4][ss] + s[27][ss]); t10 = (float)(s[4][ss] - s[27][ss]) * 0.553103896034f;
04144     t11 = (float)(s[5][ss] + s[26][ss]); t12 = (float)(s[5][ss] - s[26][ss]) * 0.582934968206f;
04145     t13 = (float)(s[6][ss] + s[25][ss]); t14 = (float)(s[6][ss] - s[25][ss]) * 0.622504123036f;
04146     t15 = (float)(s[7][ss] + s[24][ss]); t16 = (float)(s[7][ss] - s[24][ss]) * 0.674808341455f;
04147     t17 = (float)(s[8][ss] + s[23][ss]); t18 = (float)(s[8][ss] - s[23][ss]) * 0.744536271002f;
04148     t19 = (float)(s[9][ss] + s[22][ss]); t20 = (float)(s[9][ss] - s[22][ss]) * 0.839349645416f;
04149     t21 = (float)(s[10][ss] + s[21][ss]); t22 = (float)(s[10][ss] - s[21][ss]) * 0.972568237862f;
04150     t23 = (float)(s[11][ss] + s[20][ss]); t24 = (float)(s[11][ss] - s[20][ss]) * 1.16943993343f;
04151     t25 = (float)(s[12][ss] + s[19][ss]); t26 = (float)(s[12][ss] - s[19][ss]) * 1.48416461631f;
04152     t27 = (float)(s[13][ss] + s[18][ss]); t28 = (float)(s[13][ss] - s[18][ss]) * 2.05778100995f;
04153     t29 = (float)(s[14][ss] + s[17][ss]); t30 = (float)(s[14][ss] - s[17][ss]) * 3.40760841847f;
04154     t31 = (float)(s[15][ss] + s[16][ss]); t32 = (float)(s[15][ss] - s[16][ss]) * 10.1900081235f;
04155
04156     t33 = t01 + t31; t31 = (t01 - t31) * 0.502419286188f;
04157     t01 = t03 + t29; t29 = (t03 - t29) * 0.52249861494f;
04158     t03 = t05 + t27; t27 = (t05 - t27) * 0.566944034816f;
04159     t05 = t07 + t25; t25 = (t07 - t25) * 0.64682178336f;
04160     t07 = t09 + t23; t23 = (t09 - t23) * 0.788154623451f;
04161     t09 = t11 + t21; t21 = (t11 - t21) * 1.0607768599f;
04162     t11 = t13 + t19; t19 = (t13 - t19) * 1.72244709824f;
04163     t13 = t15 + t17; t17 = (t15 - t17) * 5.10114861869f;
04164     t15 = t33 + t13; t13 = (t33 - t13) * 0.509795579104f;
04165     t33 = t01 + t11; t01 = (t01 - t11) * 0.601344886935f;
04166     t11 = t03 + t09; t09 = (t03 - t09) * 0.899976223136f;
04167     t03 = t05 + t07; t07 = (t05 - t07) * 2.56291544774f;
04168     t05 = t15 + t03; t15 = (t15 - t03) * 0.541196100146f;
04169     t03 = t33 + t11; t11 = (t33 - t11) * 1.30656296488f;
04170     t33 = t05 + t03; t05 = (t05 - t03) * 0.707106781187f;
04171     t03 = t15 + t11; t15 = (t15 - t11) * 0.707106781187f;
04172     t03 += t15;
04173     t11 = t13 + t07; t13 = (t13 - t07) * 0.541196100146f;
04174     t07 = t01 + t09; t09 = (t01 - t09) * 1.30656296488f;
04175     t01 = t11 + t07; t07 = (t11 - t07) * 0.707106781187f;
04176     t11 = t13 + t09; t13 = (t13 - t09) * 0.707106781187f;
04177     t11 += t13; t01 += t11;
04178     t11 += t07; t07 += t13;
04179     t09 = t31 + t17; t31 = (t31 - t17) * 0.509795579104f;

```

```

04180     t17 = t29 + t19; t29 = (t29 - t19) * 0.601344886935f;
04181     t19 = t27 + t21; t21 = (t27 - t21) * 0.899976223136f;
04182     t27 = t25 + t23; t23 = (t25 - t23) * 2.56291544774f;
04183     t25 = t09 + t27; t09 = (t09 - t27) * 0.541196100146f;
04184     t27 = t17 + t19; t19 = (t17 - t19) * 1.30656296488f;
04185     t17 = t25 + t27; t27 = (t25 - t27) * 0.707106781187f;
04186     t25 = t09 + t19; t19 = (t09 - t19) * 0.707106781187f;
04187     t25 += t19;
04188     t09 = t31 + t23; t31 = (t31 - t23) * 0.541196100146f;
04189     t23 = t29 + t21; t21 = (t29 - t21) * 1.30656296488f;
04190     t29 = t09 + t23; t23 = (t09 - t23) * 0.707106781187f;
04191     t09 = t31 + t21; t31 = (t31 - t21) * 0.707106781187f;
04192     t09 += t31; t29 += t09; t09 += t23; t23 += t31;
04193     t17 += t29; t29 += t25; t25 += t09; t09 += t27;
04194     t27 += t23; t23 += t19; t19 += t31;
04195     t21 = t02 + t32; t02 = (t02 - t32) * 0.502419286188f;
04196     t32 = t04 + t30; t04 = (t04 - t30) * 0.52249861494f;
04197     t30 = t06 + t28; t28 = (t06 - t28) * 0.566944034816f;
04198     t06 = t08 + t26; t08 = (t08 - t26) * 0.64682178336f;
04199     t26 = t10 + t24; t10 = (t10 - t24) * 0.788154623451f;
04200     t24 = t12 + t22; t22 = (t12 - t22) * 1.06067768599f;
04201     t12 = t14 + t20; t20 = (t14 - t20) * 1.72244709824f;
04202     t14 = t16 + t18; t16 = (t16 - t18) * 5.10114861869f;
04203     t18 = t21 + t14; t14 = (t21 - t14) * 0.509795579104f;
04204     t21 = t32 + t12; t32 = (t32 - t12) * 0.601344886935f;
04205     t12 = t30 + t24; t24 = (t30 - t24) * 0.899976223136f;
04206     t30 = t06 + t26; t26 = (t06 - t26) * 2.56291544774f;
04207     t06 = t18 + t30; t18 = (t18 - t30) * 0.541196100146f;
04208     t30 = t21 + t12; t12 = (t21 - t12) * 1.30656296488f;
04209     t21 = t06 + t30; t30 = (t06 - t30) * 0.707106781187f;
04210     t06 = t18 + t12; t12 = (t18 - t12) * 0.707106781187f;
04211     t06 += t12;
04212     t18 = t14 + t26; t26 = (t14 - t26) * 0.541196100146f;
04213     t14 = t32 + t24; t24 = (t32 - t24) * 1.30656296488f;
04214     t32 = t18 + t14; t14 = (t18 - t14) * 0.707106781187f;
04215     t18 = t26 + t24; t24 = (t26 - t24) * 0.707106781187f;
04216     t18 += t24; t32 += t18;
04217     t18 += t14; t26 = t14 + t24;
04218     t14 = t02 + t16; t02 = (t02 - t16) * 0.509795579104f;
04219     t16 = t04 + t20; t04 = (t04 - t20) * 0.601344886935f;
04220     t20 = t28 + t22; t22 = (t28 - t22) * 0.899976223136f;
04221     t28 = t08 + t10; t10 = (t08 - t10) * 2.56291544774f;
04222     t08 = t14 + t28; t14 = (t14 - t28) * 0.541196100146f;
04223     t28 = t16 + t20; t20 = (t16 - t20) * 1.30656296488f;
04224     t16 = t08 + t28; t28 = (t08 - t28) * 0.707106781187f;
04225     t08 = t14 + t20; t20 = (t14 - t20) * 0.707106781187f;
04226     t08 += t20;
04227     t14 = t02 + t10; t02 = (t02 - t10) * 0.541196100146f;
04228     t10 = t04 + t22; t22 = (t04 - t22) * 1.30656296488f;
04229     t04 = t14 + t10; t10 = (t14 - t10) * 0.707106781187f;
04230     t14 = t02 + t22; t02 = (t02 - t22) * 0.707106781187f;
04231     t14 += t02; t04 += t14; t14 += t10; t10 += t02;
04232     t16 += t04; t04 += t08; t08 += t14; t14 += t28;
04233     t28 += t10; t10 += t20; t20 += t02; t21 += t16;
04234     t16 += t32; t32 += t04; t04 += t06; t06 += t08;
04235     t08 += t18; t18 += t14; t14 += t30; t30 += t28;
04236     t28 += t26; t26 += t10; t10 += t12; t12 += t20;
04237     t20 += t24; t24 += t02;
04238
04239     d[dp + 48] = -t33;
04240     d[dp + 49] = d[dp + 47] = -t21;
04241     d[dp + 50] = d[dp + 46] = -t17;
04242     d[dp + 51] = d[dp + 45] = -t16;
04243     d[dp + 52] = d[dp + 44] = -t01;
04244     d[dp + 53] = d[dp + 43] = -t32;
04245     d[dp + 54] = d[dp + 42] = -t29;
04246     d[dp + 55] = d[dp + 41] = -t04;
04247     d[dp + 56] = d[dp + 40] = -t03;
04248     d[dp + 57] = d[dp + 39] = -t06;
04249     d[dp + 58] = d[dp + 38] = -t25;
04250     d[dp + 59] = d[dp + 37] = -t08;
04251     d[dp + 60] = d[dp + 36] = -t11;
04252     d[dp + 61] = d[dp + 35] = -t18;
04253     d[dp + 62] = d[dp + 34] = -t09;
04254     d[dp + 63] = d[dp + 33] = -t14;
04255     d[dp + 32] = -t05;
04256     d[dp + 0] = t05; d[dp + 31] = -t30;
04257     d[dp + 1] = t30; d[dp + 30] = -t27;
04258     d[dp + 2] = t27; d[dp + 29] = -t28;
04259     d[dp + 3] = t28; d[dp + 28] = -t07;
04260     d[dp + 4] = t07; d[dp + 27] = -t26;
04261     d[dp + 5] = t26; d[dp + 26] = -t23;
04262     d[dp + 6] = t23; d[dp + 25] = -t10;
04263     d[dp + 7] = t10; d[dp + 24] = -t15;
04264     d[dp + 8] = t15; d[dp + 23] = -t12;
04265     d[dp + 9] = t12; d[dp + 22] = -t19;
04266     d[dp + 10] = t19; d[dp + 21] = -t20;

```

```

04267     d[dp + 11] = t20; d[dp + 20] = -t13;
04268     d[dp + 12] = t13; d[dp + 19] = -t24;
04269     d[dp + 13] = t24; d[dp + 18] = -t31;
04270     d[dp + 14] = t31; d[dp + 17] = -t02;
04271     d[dp + 15] = t02; d[dp + 16] = 0.0;
04272 }
04273
04274
04275 #endif // PL_MPEG_IMPLEMENTATION

```

6.5 video.h

```

00001 #include "../core/include/subsystems/screen.h"
00002 #include "pl_mpeg.h"
00003 #include <string>
00004
00006 void set_video(const std::string &filename);
00008 void video_restart();
00009 // plays the video set by set_video()
0010 // because of memory constraints we're limited to one video at a time
0011 class VideoPlayer : public screen::Page {
0012     public:
0013     VideoPlayer();
0014     void update(bool was_pressed, int x, int y) override;
0015
0016     void draw(vex::brain::lcd &screen, bool first_draw,
0017               unsigned int frame_number) override;
0018 };

```

6.6 layout.h

```

00001 #include <cmath>
00002 #include <functional>
00003
00004 struct SliderCfg{
00005     double &val;
00006     double min;
00007     double max;
00008 };
00009
00010
00011

```

6.7 lift.h

```

00001 #pragma once
00002
00003 #include "vex.h"
00004 #include "../core/include/utils/controls/pid.h"
00005 #include <iostream>
00006 #include <map>
00007 #include <atomic>
00008 #include <vector>
00009
0010 using namespace vex;
0011 using namespace std;
0012
0020 template <typename T>
0021 class Lift
0022 {
0023     public:
0031     struct lift_cfg_t
0032     {
0033         double up_speed, down_speed;
0034         double softstop_up, softstop_down;
0035
0036         PID::pid_config_t lift_pid_cfg;
0037     };
0038
0060     Lift(motor_group &lift_motors, lift_cfg_t &lift_cfg, map<T, double> &setpoint_map, limit
0061     *homing_switch=NULL)
0061     : lift_motors(lift_motors), cfg(lift_cfg), lift_pid(cfg.lift_pid_cfg), setpoint_map(setpoint_map),
0061     homing_switch(homing_switch)
0062     {

```

```
00063
00064     is_async = true;
00065     setpoint = 0;
00066
00067     // Create a background task that is constantly updating the lift PID, if requested.
00068     // Set once, and forget.
00069     task t([](void* ptr){
00070         Lift &lift = *((Lift*) ptr);
00071
00072         while(true)
00073         {
00074             if(lift.get_async())
00075                 lift.hold();
00076
00077             vexDelay(50);
00078         }
00079
00080         return 0;
00081     }, this);
00082
00083 }
00084
00085 void control_continuous(bool up_ctrl, bool down_ctrl)
00086 {
00087     static timer tmr;
00088
00089     double cur_pos = 0;
00090
00091     // Check if there's a hook for a custom sensor. If not, use the motors.
00092     if(get_sensor == NULL)
00093         cur_pos = lift_motors.position(rev);
00094     else
00095         cur_pos = get_sensor();
00096
00097     if(up_ctrl && cur_pos < cfg.softstop_up)
00098     {
00099         lift_motors.spin(directionType::fwd, cfg.up_speed, volt);
00100         setpoint = cur_pos + .3;
00101
00102         // std::cout << "DEBUG OUT: UP " << setpoint << ", " << tmr.time(sec) << ", " << cfg.down_speed <<
00103         "\n";
00104
00105         // Disable the PID while going UP.
00106         is_async = false;
00107     } else if(down_ctrl && cur_pos > cfg.softstop_down)
00108     {
00109         // Lower the lift slowly, at a rate defined by down_speed
00110         if(setpoint > cfg.softstop_down)
00111             setpoint = setpoint - (tmr.time(sec) * cfg.down_speed);
00112
00113         // std::cout << "DEBUG OUT: DOWN " << setpoint << ", " << tmr.time(sec) << ", " << cfg.down_speed <<
00114         "\n";
00115         is_async = true;
00116     } else
00117     {
00118         // Hold the lift at the last setpoint
00119         is_async = true;
00120     }
00121
00122     tmr.reset();
00123 }
00124
00125 void control_manual(bool up_btn, bool down_btn, int volt_up, int volt_down)
00126 {
00127     static bool down_hold = false;
00128     static bool init = true;
00129
00130     // Allow for setting position while still calling this function
00131     if(init || up_btn || down_btn)
00132     {
00133         init = false;
00134         is_async = false;
00135     }
00136
00137     double rev = lift_motors.position(rotationUnits::rev);
00138
00139     if(rev < cfg.softstop_down && down_btn)
00140         down_hold = true;
00141     else if( !down_btn )
00142         down_hold = false;
00143
00144     if(up_btn && rev < cfg.softstop_up)
00145         lift_motors.spin(directionType::fwd, volt_up, voltageUnits::volt);
00146     else if(down_btn && rev > cfg.softstop_down && !down_hold)
00147         lift_motors.spin(directionType::rev, volt_down, voltageUnits::volt);
00148     else
00149         lift_motors.spin(directionType::fwd, 0, voltageUnits::volt);
00150 }
```

```

00164    }
00165
00177 void control_setpoints(bool up_step, bool down_step, vector<T> pos_list)
00178 {
00179     // Make sure inputs are only processed on the rising edge of the button
00180     static bool up_last = up_step, down_last = down_step;
00181
00182     bool up_rising = up_step && !up_last;
00183     bool down_rising = down_step && !down_last;
00184
00185     up_last = up_step;
00186     down_last = down_step;
00187
00188     static int cur_index = 0;
00189
00190     // Avoid an index overflow. Shouldn't happen unless the user changes pos_list between calls.
00191     if(cur_index >= pos_list.size())
00192         cur_index = pos_list.size() - 1;
00193
00194     // Increment or decrement the index of the list, bringing it up or down.
00195     if(up_rising && cur_index < (pos_list.size() - 1))
00196         cur_index++;
00197     else if(down_rising && cur_index > 0)
00198         cur_index--;
00199
00200     // Set the lift to hold the position in the background with the PID loop
00201     set_position(pos_list[cur_index]);
00202     is_async = true;
00203
00204 }
00205
00214 bool set_position(T pos)
00215 {
00216     this->setpoint = setpoint_map[pos];
00217     is_async = true;
00218
00219     return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
00220 }
00221
00228 bool set_setpoint(double val)
00229 {
00230     this->setpoint = val;
00231     return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
00232 }
00233
00237 double get_setpoint()
00238 {
00239     return this->setpoint;
00240 }
00241
00246 void hold()
00247 {
00248     lift_pid.set_target(setpoint);
00249     // std::cout << "DEBUG OUT: SETPOINT " << setpoint << "\n";
00250
00251     if(get_sensor != NULL)
00252         lift_pid.update(get_sensor());
00253     else
00254         lift_pid.update(lift_motors.position(rev));
00255
00256     // std::cout << "DEBUG OUT: ROTATION " << lift_motors.rotation(rev) << "\n\n";
00257
00258     lift_motors.spin(fwd, lift_pid.get(), volt);
00259 }
00260
00265 void home()
00266 {
00267     static timer tmr;
00268     tmr.reset();
00269
00270     while(tmr.time(sec) < 3)
00271     {
00272         lift_motors.spin(directionType::rev, 6, volt);
00273
00274         if (homing_switch == NULL && lift_motors.current(currentUnits::amp) > 1.5)
00275             break;
00276         else if (homing_switch != NULL && homing_switch->pressing())
00277             break;
00278     }
00279
00280     if(reset_sensor != NULL)
00281         reset_sensor();
00282
00283     lift_motors.resetPosition();
00284     lift_motors.stop();
00285
00286 }
```

```

00287
00289     bool get_async()
00290     {
00291         return is_async;
00292     }
00293
00294
00295     void set_async(bool val)
00296     {
00297         this->is_async = val;
00298     }
00299
00300
00301     void set_sensor_function(double (*fn_ptr) (void))
00302     {
00303         this->get_sensor = fn_ptr;
00304     }
00305
00306
00307     void set_sensor_reset(void (*fn_ptr) (void))
00308     {
00309         this->reset_sensor = fn_ptr;
00310     }
00311
00312     private:
00313
00314     motor_group &lift_motors;
00315     lift_cfg_t &cfg;
00316     PID lift_pid;
00317     map<T, double> &setpoint_map;
00318     limit *homing_switch;
00319
00320     atomic<double> setpoint;
00321     atomic<bool> is_async;
00322
00323     double (*get_sensor) (void) = NULL;
00324     void (*reset_sensor) (void) = NULL;
00325
00326
00327     00346 };

```

6.8 mecanum_drive.h

```

00001 #pragma once
00002
00003 #include "vex.h"
00004 #include "../core/include/utils/controls/pid.h"
00005
00006 #ifndef PI
00007 #define PI 3.141592654
00008 #endif
00009
00010
00011 class MecanumDrive
00012 {
00013
00014     public:
00015
00016     struct mecanumdrive_config_t
00017     {
00018         // PID configurations for autonomous driving
00019         PID::pid_config_t drive_pid_conf;
00020         PID::pid_config_t drive_gyro_pid_conf;
00021         PID::pid_config_t turn_pid_conf;
00022
00023         // Diameter of the mecanum wheels
00024         double drive_wheel_diam;
00025
00026         // Diameter of the perpendicular undriven encoder wheel
00027         double lateral_wheel_diam;
00028
00029         // Width between the center of the left and right wheels
00030         double wheelbase_width;
00031
00032     };
00033
00034     MecanumDrive(vex::motor &left_front, vex::motor &right_front, vex::motor &left_rear, vex::motor
00035     &right_rear,
00036             vex::rotation *lateral_wheel=NULL, vex::inertial *imu=NULL, mecanumdrive_config_t
00037     *config=NULL);
00038
00039     void drive_raw(double direction_deg, double magnitude, double rotation);
00040
00041     void drive(double left_y, double left_x, double right_x, int power=2);
00042
00043     bool auto_drive(double inches, double direction, double speed, bool gyro_correction=true);
00044
00045
00046
00047
00048
00049
00050
00051
00052
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
00076
00077
00078
00079
00080
00081

```

```

00092     bool auto_turn(double degrees, double speed, bool ignore_imu=false);
00093
00094 private:
00095
00096     vex::motor &left_front, &right_front, &left_rear, &right_rear;
00097
00098     mecanumdrive_config_t *config;
00099     vex::rotation *lateral_wheel;
00100     vex::inertial *imu;
00101
00102     PID *drive_pid = NULL;
00103     PID *drive_gyro_pid = NULL;
00104     PID *turn_pid = NULL;
00105
00106     bool init = true;
00107
00108 };

```

6.9 odometry_3wheel.h

```

00001 #pragma once
00002 #include "../core/include/subsystems/odometry/odometry_base.h"
00003 #include "../core/include/subsystems/tank_drive.h"
00004 #include "../core/include/subsystems/custom_encoder.h"
00005
00032 class Odometry3Wheel : public OdometryBase
00033 {
00034     public:
00035
00040     typedef struct
00041     {
00042         double wheelbase_dist;
00043         double off_axis_center_dist;
00044         double wheel_diam;
00046     } odometry3wheel_cfg_t;
00047
00057     Odometry3Wheel(CustomEncoder &lside_fwd, CustomEncoder &rside_fwd, CustomEncoder &off_axis,
00058     odometry3wheel_cfg_t &cfg, bool is_async=true);
00058
00065     pose_t update() override;
00066
00075     void tune(vex::controller &con, TankDrive &drive);
00076
00077     private:
00078
00091     static pose_t calculate_new_pos(double lside_delta_deg, double rside_delta_deg, double
00092     offax_delta_deg, pose_t old_pos, odometry3wheel_cfg_t cfg);
00092
00093     CustomEncoder &lside_fwd, &rside_fwd, &off_axis;
00094     odometry3wheel_cfg_t &cfg;
00095
00096
00097 };

```

6.10 odometry_base.h

```

00001 #pragma once
00002
00003 #include "vex.h"
00004 #include "../core/include/utils/geometry.h"
00005 #include "../core/include/robot_specs.h"
00006 #include "../core/include/utils/command_structure/auto_command.h"
00007
00008 #ifndef PI
00009 #define PI 3.141592654
00010 #endif
00011
00012
00013
00026 class OdometryBase
00027 {
00028     public:
00029
00035     OdometryBase(bool is_async);
00036
00041     pose_t get_position(void);
00042
00047     virtual void set_position(const pose_t& newpos=zero_pos);
00048     AutoCommand *SetPositionCmd(const pose_t &newpos=zero_pos);

```

```

00053     virtual pose_t update() = 0;
00054
00062     static int background_task(void* ptr);
00063
00069     void end_async();
00070
00077     static double pos_diff(pose_t start_pos, pose_t end_pos);
00078
00085     static double rot_diff(pose_t pos1, pose_t pos2);
00086
00095     static double smallest_angle(double start_deg, double end_deg);
00096
00098     bool end_task = false;
00099
00104     double get_speed();
00105
00110     double get_accel();
00111
00116     double get_angular_speed_deg();
00117
00122     double get_angular_accel_deg();
00123
00127     inline static constexpr pose_t zero_pos = { .x=0.0L, .y=0.0L, .rot=90.0L };
00128
00129 protected:
00133     vex::task *handle;
00134
00138     vex::mutex mut;
00139
00143     pose_t current_pos;
00144
00145     double speed;
00146     double accel;
00147     double ang_speed_deg;
00148     double ang_accel_deg;
00149 };

```

6.11 odometry_tank.h

```

00001 #pragma once
00002
00003 #include "../core/include/subsystems/odometry/odometry_base.h"
00004 #include "../core/include/subsystems/custom_encoder.h"
00005 #include "../core/include/utils/geometry.h"
00006 #include "../core/include/utils/vector2d.h"
00007 #include "../core/include/utils/moving_average.h"
00008
00009 #include "../core/include/robot_specs.h"
00010
00011 static int background_task(void* odom_obj);
00012
00013
00020 class OdometryTank : public OdometryBase
00021 {
00022 public:
00031     OdometryTank(vex::motor_group &left_side, vex::motor_group &right_side, robot_specs_t &config,
vex::inertial *imu=NULL, bool is_async=true);
00032
00042     OdometryTank(CustomEncoder &left_custom_enc, CustomEncoder &right_custom_enc, robot_specs_t
&config, vex::inertial *imu=NULL, bool is_async=true);
00043
00053     OdometryTank(vex::encoder &left_vex_enc, vex::encoder &right_vex_enc, robot_specs_t &config,
vex::inertial *imu=NULL, bool is_async=true);
00054
00059     pose_t update() override;
00060
00065     void set_position(const pose_t &newpos=zero_pos) override;
00066
00067
00068
00069 private:
00073     static pose_t calculate_new_pos(robot_specs_t &config, pose_t &stored_info, double lside_diff,
double rside_diff, double angle_deg);
00074
00075     vex::motor_group *left_side, *right_side;
00076     CustomEncoder *left_custom_enc, *right_custom_enc;
00077     vex::encoder *left_vex_enc, *right_vex_enc;
00078     vex::inertial *imu;
00079     robot_specs_t &config;
00080
00081     double rotation_offset = 0;
00082     ExponentialMovingAverage ema = ExponentialMovingAverage(3);
00083
00084 };

```

6.12 screen.h

```

00001 #pragma once
00002 #include "vex.h"
00003 #include <vector>
00004 #include <functional>
00005 #include <map>
00006 #include <cassert>
00007 #include "../core/include/subsystems/odometry/odometry_base.h"
00008 #include "../core/include/utils/graph_drawer.h"
00009 #include "../core/include/utils/controls/pid.h"
0010 #include "../core/include/utils/controls/pidff.h"
0011
0012 namespace screen
0013 {
0015     class ButtonWidget
0016     {
0017         public:
0022         ButtonWidget(std::function<void(void)> onpress, Rect rect, std::string name) :
0023             onpress(onpress), rect(rect), name(name) {}
0027         ButtonWidget(void (*onpress)(), Rect rect, std::string name) : onpress(onpress), rect(rect),
0028             name(name) {}
0029         bool update(bool was_pressed, int x, int y);
0030         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
0031
0032         private:
0033             std::function<void(void)> onpress;
0034             Rect rect;
0035             std::string name = "";
0036             bool was_pressed_last = false;
0037     };
0038
0039     class SliderWidget
0040     {
0041         public:
0045         SliderWidget(double &val, double low, double high, Rect rect, std::string name) : value(val),
0046             low(low), high(high), rect(rect), name(name) {}
0047
0048         bool update(bool was_pressed, int x, int y);
0049         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
0050
0051         private:
0052             double &value;
0053
0054             double low;
0055             double high;
0056
0057             Rect rect;
0058             std::string name = "";
0059     };
0060
0061     struct WidgetConfig;
0062
0063     struct SliderConfig
0064     {
0065         double &val;
0066         double low;
0067         double high;
0068     };
0069     struct ButtonConfig
0070     {
0071         std::function<void()> onclick;
0072     };
0073     struct CheckboxConfig
0074     {
0075         std::function<void(bool)> onupdate;
0076     };
0077     struct LabelConfig
0078     {
0079         std::string label;
0080     };
0081     struct TextConfig
0082     {
0083         std::function<std::string()> text;
0084     };
0085     struct SizedWidget
0086     {
0087         int size;
0088         WidgetConfig &widget;
0089     };
0090     struct WidgetConfig
0091     {
0092         enum Type
0093         {
0094             Col,
0095             Row,
0096             Grid
0097         };
0098         std::vector<WidgetConfig> children;
0099     };
0100 }
0101
0102
0103
0104
0105
0106
0107
0108
0109
0110

```

```

00111         Row,
00112         Slider,
00113         Button,
00114         Checkbox,
00115         Label,
00116         Text,
00117         Graph,
00118     };
00119     Type type;
00120     union
00121     {
00122         std::vector<SizedWidget> widgets;
00123         SliderConfig slider;
00124         ButtonConfig button;
00125         CheckboxConfig checkbox;
00126         LabelConfig label;
00127         TextConfig text;
00128         GraphDrawer *graph;
00129     } config;
00130 };
00131
00132 class Page;
00133 class Page
00134 {
00135 public:
00136     virtual void update(bool was_pressed, int x, int y);
00137     virtual void draw(vex::brain::lcd &screen, bool first_draw,
00138                       unsigned int frame_number);
00139 };
00140
00141 struct ScreenRect
00142 {
00143     uint32_t x1;
00144     uint32_t y1;
00145     uint32_t x2;
00146     uint32_t y2;
00147 };
00148 void draw_widget(WidgetConfig &widget, ScreenRect rect);
00149
00150 class WidgetPage : public Page
00151 {
00152 public:
00153     WidgetPage(WidgetConfig &cfg) : base_widget(cfg) {}
00154     void update(bool was_pressed, int x, int y) override;
00155
00156     void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override
00157     {
00158         draw_widget(base_widget, {.x1 = 20, .y1 = 0, .x2 = 440, .y2 = 240});
00159     }
00160
00161 private:
00162     WidgetConfig &base_widget;
00163 };
00164
00165 void start_screen(vex::brain::lcd &screen, std::vector<Page *> pages, int first_page = 0);
00166
00167 void next_page();
00168 void prev_page();
00169 void goto_page(size_t page);
00170
00171 void stop_screen();
00172
00173 using update_func_t = std::function<void(bool, int, int)>;
00174
00175 using draw_func_t = std::function<void(vex::brain::lcd &screen, bool, unsigned int)>;
00176
00177 class StatsPage : public Page
00178 {
00179 public:
00180     StatsPage(std::map<std::string, vex::motor &> motors);
00181     void update(bool was_pressed, int x, int y) override;
00182     void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00183
00184 private:
00185     void draw_motor_stats(const std::string &name, vex::motor &mot, unsigned int frame, int x, int
00186 y, vex::brain::lcd &scr);
00187     std::map<std::string, vex::motor &> motors;
00188     static const int y_start = 0;
00189     static const int per_column = 4;
00190     static const int row_height = 20;
00191     static const int row_width = 200;
00192 };
00193
00194 class OdometryPage : public Page
00195 {

```

```

00230     public:
00231         OdometryPage(OdometryBase &odom, double robot_width, double robot_height, bool do_trail);
00232         void update(bool was_pressed, int x, int y) override;
00233         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00234
00235     private:
00236         static const int path_len = 40;
00237         static constexpr char const *field_filename = "vex_field_240p.png";
00238
00239         OdometryBase &odom;
00240         double robot_width;
00241         double robot_height;
00242         uint8_t *buf = nullptr;
00243         int buf_size = 0;
00244         pose_t path[path_len];
00245         int path_index = 0;
00246         bool do_trail;
00247         GraphDrawer velocity_graph;
00248     };
00249
00250     class FunctionPage : public Page
00251     {
00252     public:
00253         FunctionPage(update_func_t update_f, draw_func_t draw_t);
00254         void update(bool was_pressed, int x, int y) override;
00255         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00256
00257     private:
00258         update_func_t update_f;
00259         draw_func_t draw_f;
00260     };
00261
00262     class PIDPage : public Page
00263     {
00264     public:
00265         PIDPage(
00266             PID &pid, std::string name, std::function<void(void)> onchange = []() {});
00267         PIDPage(
00268             PIDFF &pidff, std::string name, std::function<void(void)> onchange = []() {});
00269
00270         void update(bool was_pressed, int x, int y) override;
00271         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00272
00273     private:
00274         void zero_d_f() { cfg.d = 0; }
00275         void zero_i_f() { cfg.i = 0; }
00276
00277         PID::pid_config_t &cfg;
00278         PID &pid;
00279         const std::string name;
00280         std::function<void(void)> onchange;
00281
00282         SliderWidget p_slider;
00283         SliderWidget i_slider;
00284         SliderWidget d_slider;
00285         ButtonWidget zero_i;
00286         ButtonWidget zero_d;
00287
00288         GraphDrawer graph;
00289     };
00290
00291 };
00292
00293 }
```

6.13 tank_drive.h

```

00001 #pragma once
00002
00003 #ifndef PI
00004 #define PI 3.141592654
00005 #endif
00006
00007 #include "../core/include/robot_specs.h"
00008 #include "../core/include/subsystems/odometry/odometry_tank.h"
00009 #include "../core/include/utils/command_structure/auto_command.h"
00010 #include "../core/include/utils/controls/feedback_base.h"
00011 #include "../core/include/utils/controls/pid.h"
00012 #include "../core/include/utils/pure_pursuit.h"
00013 #include "vex.h"
00014 #include <vector>
00015
00016 using namespace vex;
00017
00018 class TankDrive {
```

```

00024     public:
00025         enum class BrakeType {
00026             None,
00027             ZeroVelocity,
00028             Smart,
00029         };
00031     };
00041     TankDrive(motor_group &left_motors, motor_group &right_motors,
00042                 robot_specs_t &config, OdometryBase *odom = NULL);
00043
00044     AutoCommand *DriveToPointCmd(point_t pt,
00045                                     vex::directionType dir = vex::forward,
00046                                     double max_speed = 1.0,
00047                                     double end_speed = 0.0);
00048     AutoCommand *DriveToPointCmd(Feedback &fb, point_t pt,
00049                                     vex::directionType dir = vex::forward,
00050                                     double max_speed = 1.0,
00051                                     double end_speed = 0.0);
00052
00053     AutoCommand *DriveForwardCmd(double dist,
00054                                     vex::directionType dir = vex::forward,
00055                                     double max_speed = 1.0,
00056                                     double end_speed = 0.0);
00057     AutoCommand *DriveForwardCmd(Feedback &fb, double dist,
00058                                     vex::directionType dir = vex::forward,
00059                                     double max_speed = 1.0,
00060                                     double end_speed = 0.0);
00061
00062     AutoCommand *TurnToHeadingCmd(double heading, double max_speed = 1.0,
00063                                     double end_speed = 0.0);
00064     AutoCommand *TurnToHeadingCmd(Feedback &fb, double heading,
00065                                     double max_speed = 1.0,
00066                                     double end_speed = 0.0);
00067
00068     AutoCommand *
00069     TurnToPointCmd(double x, double y,
00070                     vex::directionType dir = vex::directionType::fwd,
00071                     double max_speed = 1.0, double end_speed = 0.0);
00072
00073     AutoCommand *TurnDegreesCmd(double degrees, double max_speed = 1.0,
00074                                 double start_speed = 0.0);
00075     AutoCommand *TurnDegreesCmd(Feedback &fb, double degrees,
00076                                 double max_speed = 1.0, double end_speed = 0.0);
00077
00078     AutoCommand *PurePursuitCmd(PurePursuit::Path path, directionType dir,
00079                                 double max_speed = 1, double end_speed = 0);
00080     AutoCommand *PurePursuitCmd(Feedback &feedback, PurePursuit::Path path,
00081                                 directionType dir, double max_speed = 1,
00082                                 double end_speed = 0);
00083     Condition *DriveStalledCondition(double stall_time);
00084     AutoCommand *DriveTankCmd(double left, double right);
00085
00086     void stop();
00087
00088     void drive_tank(double left, double right, int power = 1,
00089                      BrakeType bt = BrakeType::None);
00090     void drive_tank_raw(double left, double right);
00091
00092     void drive_arcade(double forward_back, double left_right, int power = 1,
00093                        BrakeType bt = BrakeType::None);
00094
00095     bool drive_forward(double inches, directionType dir, Feedback &feedback,
00096                         double max_speed = 1, double end_speed = 0);
00097
00098     bool drive_forward(double inches, directionType dir, double max_speed = 1,
00099                         double end_speed = 0);
00100
00101     bool turn_degrees(double degrees, Feedback &feedback, double max_speed = 1,
00102                       double end_speed = 0);
00103
00104     bool turn_degrees(double degrees, double max_speed = 1,
00105                       double end_speed = 0);
00106
00107     bool drive_to_point(double x, double y, vex::directionType dir,
00108                          Feedback &feedback, double max_speed = 1,
00109                          double end_speed = 0);
00110
00111     bool drive_to_point(double x, double y, vex::directionType dir,
00112                          double max_speed = 1, double end_speed = 0);
00113
00114     bool turn_to_heading(double heading_deg, Feedback &feedback,
00115                           double max_speed = 1, double end_speed = 0);
00116     bool turn_to_heading(double heading_deg, double max_speed = 1,
00117                           double end_speed = 0);
00118
00119     void reset_auto();
00120
00121     static double modify_inputs(double input, int power = 2);

```

```

00268
00269     bool pure_pursuit(PurePursuit::Path path, directionType dir,
00270                         Feedback &feedback, double max_speed = 1,
00271                         double end_speed = 0);
00272
00273     bool pure_pursuit(PurePursuit::Path path, directionType dir,
00274                         double max_speed = 1, double end_speed = 0);
00275
00276 private:
00277     motor_group &left_motors;
00278     motor_group &right_motors;
00279
00280     PID correction_pid;
00281     Feedback *drive_default_feedback =
00282         NULL;
00283     Feedback *turn_default_feedback =
00284         NULL;
00285
00286     OdometryBase *odometry;
00287
00288     robot_specs_t &config;
00289
00290     bool func_initialized =
00291         false;
00292     bool is_pure_pursuit =
00293         false;
00294 };

```

6.14 auto_chooser.h

```

00001 #pragma once
00002 #include "vex.h"
00003 #include <string>
00004 #include <vector>
00005 #include "../core/include/subsystems/screen.h"
00006 #include "../core/include/utils/geometry.h"
00007
00016 class AutoChooser : public screen::Page
00017 {
00018     public:
00024     AutoChooser(std::vector<std::string> paths, size_t def = 0);
00025
00026     void update(bool was_pressed, int x, int y);
00027     void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
00028
00033     size_t get_choice();
00034
00035     protected:
00039     struct entry_t
00040     {
00041         Rect rect;
00042         std::string name;
00043     };
00044
00045     static const size_t width = 380;
00046     static const size_t height = 220;
00047
00048     size_t choice;
00049     std::vector<entry_t> list ;
00050 };

```

6.15 auto_command.h

```

00001
00007 #pragma once
00008
00009 #include "vex.h"
00010 #include <functional>
00011 #include <vector>
00012 #include <queue>
00013 #include <atomic>
00014
00015
00025 class Condition
00026 {
00027     public:
00028     Condition *Or(Condition *b);
00029     Condition *And(Condition *b);
00030     virtual bool test() = 0;

```

```
00031 };
00032
00033
00034 class AutoCommand
00035 {
00036 public:
00037     static constexpr double default_timeout = 10.0;
00038     virtual bool run() { return true; }
00039     virtual void on_timeout() {}
00040     AutoCommand *withTimeout(double t_seconds)
00041     {
00042         if (this->timeout_seconds < 0)
00043         {
00044             // should never be timed out
00045             return this;
00046         }
00047         this->timeout_seconds = t_seconds;
00048         return this;
00049     }
00050     AutoCommand *withCancelCondition(Condition *true_to_end)
00051     {
00052         this->true_to_end = true_to_end;
00053         return this;
00054     }
00055     double timeout_seconds = default_timeout;
00056     Condition *true_to_end = nullptr;
00057 };
00058
00059 class FunctionCommand : public AutoCommand
00060 {
00061 public:
00062     FunctionCommand(std::function<bool(void)> f) : f(f) {}
00063     bool run()
00064     {
00065         return f();
00066     }
00067
00068 private:
00069     std::function<bool(void)> f;
00070 };
00071
00072 // Times tested 3
00073 // Test 1 -> false
00074 // Test 2 -> false
00075 // Test 3 -> true
00076 // Returns false until the Nth time that it is called
00077 // This is pretty much only good for implementing RepeatUntil
00078 class TimesTestedCondition : public Condition
00079 {
00080 public:
00081     TimesTestedCondition(size_t N) : max(N) {}
00082     bool test() override
00083     {
00084         count++;
00085         if (count >= max)
00086         {
00087             return true;
00088         }
00089         return false;
00090     }
00091
00092 private:
00093     size_t count = 0;
00094     size_t max;
00095 };
00096
00097 class FunctionCondition : public Condition
00098 {
00099 public:
00100     FunctionCondition(
00101         std::function<bool(void)> cond, std::function<void(void)> timeout = []() {}) : cond(cond),
00102         timeout(timeout)
00103     {
00104     }
00105     bool test() override;
00106
00107 private:
00108     std::function<bool(void)> cond;
00109     std::function<void(void)> timeout;
00110 };
00111
00112 class IfTimePassed : public Condition
00113 {
00114 public:
00115     IfTimePassed(double time_s);
00116     bool test() override;
00117
00118 private:
```

```
00140     double time_s;
00141     vex::timer tmr;
00142 };
00143
00145 class WaitUntilCondition : public AutoCommand
00146 {
00147 public:
00148     WaitUntilCondition(Condition *cond) : cond(cond) {}
00149     bool run() override
00150     {
00151         return cond->test();
00152     }
00153
00154 private:
00155     Condition *cond;
00156 };
00157
00160
00163 class InOrder : public AutoCommand
00164 {
00165 public:
00166     InOrder(const InOrder &other) = default;
00167     InOrder(std::queue<AutoCommand *> cmdqs);
00168     InOrder(std::initializer_list<AutoCommand *> cmdqs);
00169     bool run() override;
00170     void on_timeout() override;
00171
00172 private:
00173     AutoCommand *current_command = nullptr;
00174     std::queue<AutoCommand *> cmdqs;
00175     vex::timer tmr;
00176 };
00177
00180 class Parallel : public AutoCommand
00181 {
00182 public:
00183     Parallel(std::initializer_list<AutoCommand *> cmdqs);
00184     bool run() override;
00185     void on_timeout() override;
00186
00187 private:
00188     std::vector<AutoCommand *> cmdqs;
00189     std::vector<vex::task *> runners;
00190 };
00191
00195 class Branch : public AutoCommand
00196 {
00197 public:
00198     Branch(Condition *cond, AutoCommand *false_choice, AutoCommand *true_choice);
00199     ~Branch();
00200     bool run() override;
00201     void on_timeout() override;
00202
00203 private:
00204     AutoCommand *false_choice;
00205     AutoCommand *true_choice;
00206     Condition *cond;
00207     bool choice = false;
00208     bool chosen = false;
00209     vex::timer tmr;
00210 };
00211
00215 class Async : public AutoCommand
00216 {
00217 public:
00218     Async(AutoCommand *cmd) : cmd(cmd) {}
00219     bool run() override;
00220
00221 private:
00222     AutoCommand *cmd = nullptr;
00223 };
00224
00225 class RepeatUntil : public AutoCommand
00226 {
00227 public:
00228     RepeatUntil(InOrder cmdqs, size_t repeats);
00229     RepeatUntil(InOrder cmdqs, Condition *true_to_end);
00230     bool run() override;
00231     void on_timeout() override;
00232
00233 private:
00234     const InOrder cmdqs;
00235     InOrder *working_cmds;
00236     Condition *cond;
00237 }
```

6.16 basic_command.h

```

00001
00014 #pragma once
00015
00016 #include "../core/include/utils/command_structure/auto_command.h"
00017
00018 //Basic Motor Classes-----
00019
00024 class BasicSpinCommand : public AutoCommand {
00025     public:
00026
00027     //Enumerator for the type of power setting in the motor
00028     enum type {percent,voltage,velocity};
00029
00038     BasicSpinCommand(vex::motor &motor, vex::directionType dir, BasicSpinCommand::type setting,
00039                         double power);
00039
00046     bool run() override;
00047
00048     private:
00049
00050     vex::motor &motor;
00051
00052     type setting;
00053
00054     vex::directionType dir;
00055
00056     double power;
00057 };
00062 class BasicStopCommand : public AutoCommand{
00063     public:
00064
00071     BasicStopCommand(vex::motor &motor, vex::brakeType setting);
00072
00079     bool run() override;
00080
00081     private:
00082
00083     vex::motor &motor;
00084
00085     vex::brakeType setting;
00086 };
00087
00088 //Basic Solenoid Commands-----
00089
00094 class BasicSolenoidSet : public AutoCommand{
00095     public:
00096
00103     BasicSolenoidSet(vex::pneumatics &solenoid, bool setting);
00104
00111     bool run() override;
00112
00113     private:
00114
00115     vex::pneumatics &solenoid;
00116
00117     bool setting;
00118 };

```

6.17 command_controller.h

```

00001
00010 #pragma once
00011 #include "../core/include/utils/command_structure/auto_command.h"
00012 #include <queue>
00013 #include <vector>
00014
00015 class CommandController {
00016     public:
00019     [[deprecated("Empty constructor is bad. Use list constructor "
00020                 "instead.")]] CommandController()
00021         : command_queue({}) {}
00022
00026     CommandController(std::initializer_list<AutoCommand *> cmd)
00027         : command_queue(cmd) {}
00035     [[deprecated("Use list constructor instead. If you need to make a decision "
00036                 "before adding new commands, use Branch "
00037                 "(https://github.com/RIT-VEX-U/Core/wiki/"
00038                 "%3-%7C-Utilites#commandcontroller)")]] void
00039     add(std::vector<AutoCommand *> cmd);
00040     void add(AutoCommand *cmd, double timeout_seconds = 10.0);
00041

```

```

00053     [[deprecated("Use list constructor instead. If you need to make a decision "
00054             "before adding new commands, use Branch "
00055             "(https://github.com/RIT-VEX-U/Core/wiki/"
00056             "3-%7C-Utilites#commandcontroller)")]
00057     void add(std::vector<AutoCommand *> cmd, double timeout_sec);
00058     void add_delay(int ms);
00059
00060     void add_cancel_func(std::function<bool(void)> true_if_cancel);
00061
00062     void run();
00063
00064     bool last_command_timed_out();
00065
00066     private:
00067     std::queue<AutoCommand *> command_queue;
00068     bool command_timed_out = false;
00069     std::function<bool()> should_cancel = []() { return false; };
00070
00071 };

```

6.18 delay_command.h

```

00001
00002 #pragma once
00003
00004
00005 #include "../core/include/utils/command_structure/auto_command.h"
00006
00007 class DelayCommand: public AutoCommand {
00008     public:
00009         DelayCommand(int ms): ms(ms) {}
00010
00011         bool run() override {
00012             vexDelay(ms);
00013             return true;
00014         }
00015
00016     private:
00017         // amount of milliseconds to wait
00018         int ms;
00019
00020 };

```

6.19 drive_commands.h

```

00001
00002 #pragma once
00003
00004
00005 #include "vex.h"
00006 #include "../core/include/utils/geometry.h"
00007 #include "../core/include/utils/command_structure/auto_command.h"
00008 #include "../core/include/subsystems/tank_drive.h"
00009
00010 using namespace vex;
00011
00012 // ===== DRIVING =====
00013
00014 class DriveForwardCommand: public AutoCommand
00015 {
00016     public:
00017         DriveForwardCommand(TankDrive &drive_sys, Feedback &feedback, double inches, directionType dir,
00018         double max_speed=1, double end_speed=0);
00019
00020         bool run() override;
00021         void on_timeout() override;
00022
00023     private:
00024         // drive system to run the function on
00025         TankDrive &drive_sys;
00026
00027         // feedback controller to use
00028         Feedback &feedback;
00029
00030         // parameters for drive_forward
00031         double inches;
00032         directionType dir;
00033         double max_speed;
00034         double end_speed;
00035
00036 };
00037
00038 class TurnDegreesCommand: public AutoCommand

```

```
00071 {
00072     public:
00073         TurnDegreesCommand(TankDrive &drive_sys, Feedback &feedback, double degrees, double max_speed = 1,
00074         double end_speed = 0);
00074
00080     bool run() override;
00084     void on_timeout() override;
00085
00086
00087     private:
00088         // drive system to run the function on
00089         TankDrive &drive_sys;
00090
00091         // feedback controller to use
00092         Feedback &feedback;
00093
00094         // parameters for turn_degrees
00095         double degrees;
00096         double max_speed;
00097         double end_speed;
00098    };
00099
00104 class DriveToPointCommand: public AutoCommand
00105 {
00106     public:
00107         DriveToPointCommand(TankDrive &drive_sys, Feedback &feedback, double x, double y, directionType
00108         dir, double max_speed = 1, double end_speed = 0);
00108         DriveToPointCommand(TankDrive &drive_sys, Feedback &feedback, point_t point, directionType dir,
00109         double max_speed=1, double end_speed = 0);
00109
00115     bool run() override;
00116
00117     private:
00118         // drive system to run the function on
00119         TankDrive &drive_sys;
00120
00124     void on_timeout() override;
00125
00126
00127         // feedback controller to use
00128         Feedback &feedback;
00129
00130         // parameters for drive_to_point
00131         double x;
00132         double y;
00133         directionType dir;
00134         double max_speed;
00135         double end_speed;
00136
00137    };
00138
00144 class TurnToHeadingCommand: public AutoCommand
00145 {
00146     public:
00147         TurnToHeadingCommand(TankDrive &drive_sys, Feedback &feedback, double heading_deg, double speed =
00148         1, double end_speed = 0);
00148
00154     bool run() override;
00158     void on_timeout() override;
00159
00160
00161     private:
00162         // drive system to run the function on
00163         TankDrive &drive_sys;
00164
00165         // feedback controller to use
00166         Feedback &feedback;
00167
00168         // parameters for turn_to_heading
00169         double heading_deg;
00170         double max_speed;
00171         double end_speed;
00172    };
00173
00177 class PurePursuitCommand: public AutoCommand
00178 {
00179     public:
00188     PurePursuitCommand(TankDrive &drive_sys, Feedback &feedback, PurePursuit::Path path, directionType
00189         dir, double max_speed=1, double end_speed=0);
00189
00193     bool run() override;
00194
00198     void on_timeout() override;
00199
00200     private:
00201         TankDrive &drive_sys;
00202         PurePursuit::Path path;
```

```

00203     directionType dir;
00204     Feedback &feedback;
00205     double max_speed;
00206     double end_speed;
00207 }
00209
00214 class DriveStopCommand: public AutoCommand
00215 {
00216     public:
00217     DriveStopCommand(TankDrive &drive_sys);
00218
00219     bool run() override;
00220     void on_timeout() override;
00221
00222     private:
00223     // drive system to run the function on
00224     TankDrive &drive_sys;
00225 };
00226
00227 // ===== ODOMETRY =====
00228
00229 class OdomSetPosition: public AutoCommand
00230 {
00231     public:
00232     OdomSetPosition(OdometryBase &odom, const pose_t &newpos=OdometryBase::zero_pos);
00233
00234     bool run() override;
00235
00236     private:
00237     // drive system with an odometry config
00238     OdometryBase &odom;
00239     pose_t newpos;
00240 };

```

6.20 flywheel_commands.h

```

00001
00007 #pragma once
00008
00009 #include "../core/include/subsystems/flywheel.h"
00010 #include "../core/include/utils/command_structure/auto_command.h"
00011
00017 class SpinRPMCommand: public AutoCommand {
00018     public:
00024     SpinRPMCommand(Flywheel &flywheel, int rpm);
00025
00031     bool run() override;
00032
00033     private:
00034     // Flywheel instance to run the function on
00035     Flywheel &flywheel;
00036
00037     // parameters for spin_rpm
00038     int rpm;
00039 };
00040
00045 class WaitUntilUpToSpeedCommand: public AutoCommand {
00046     public:
00052     WaitUntilUpToSpeedCommand(Flywheel &flywheel, int threshold_rpm);
00053
00059     bool run() override;
00060
00061     private:
00062     // Flywheel instance to run the function on
00063     Flywheel &flywheel;
00064
00065     // if the actual speed is equal to the desired speed +/- this value, we are ready to fire
00066     int threshold_rpm;
00067 };
00068
00074 class FlywheelStopCommand: public AutoCommand {
00075     public:
00080     FlywheelStopCommand(Flywheel &flywheel);
00081
00087     bool run() override;
00088
00089     private:
00090     // Flywheel instance to run the function on
00091     Flywheel &flywheel;
00092 };
00093

```

```

00099 class FlywheelStopMotorsCommand: public AutoCommand {
00100     public:
00105     FlywheelStopMotorsCommand(Flywheel &flywheel);
00106
00112     bool run() override;
00113
00114     private:
00115     // Flywheel instance to run the function on
00116     Flywheel &flywheel;
00117 };
00118
00124 class FlywheelStopNonTasksCommand: public AutoCommand {
00125     FlywheelStopNonTasksCommand(Flywheel &flywheel);
00126
00132     bool run() override;
00133
00134     private:
00135     // Flywheel instance to run the function on
00136     Flywheel &flywheel;
00137 };

```

6.21 bang_bang.h

```

00001 #include "../core/include/utils/controls/feedback_base.h"
00002
00003 class BangBang : public Feedback
00004 {
00005
00006     public:
00007     BangBang(double threshold, double low, double high);
00016     void init(double start_pt, double set_pt, double start_vel [[maybe_unused]] = 0.0, double end_vel
00017     [[maybe_unused]] = 0.0) override;
00018
00024     double update(double val) override;
00025
00029     double get() override;
00030
00037     void set_limits(double lower, double upper) override;
00038
00042     bool is_on_target() override;
00043
00044     private:
00045     double setpt;
00046     double sensor_val;
00047     double lower_bound, upper_bound;
00048     double last_output;
00049     double threshold;
00050 };

```

6.22 feedback_base.h

```

00001 #pragma once
00002
00010 class Feedback
00011 {
00012     public:
00021     virtual void init(double start_pt, double set_pt, double start_vel = 0.0, double end_vel = 0.0) =
0;
00022
00029     virtual double update(double val) = 0;
00030
00034     virtual double get() = 0;
00035
00042     virtual void set_limits(double lower, double upper) = 0;
00043
00047     virtual bool is_on_target() = 0;
00048
00049
00050 };

```

6.23 feedforward.h

```

00001 #pragma once
00002
00003 #include <math.h>

```

```

00004 #include <vector>
00005 #include "../core/include/utils/math_util.h"
00006 #include "../core/include/utils/moving_average.h"
00007 #include "vex.h"
00008
00029 class FeedForward
00030 {
00031     public:
00032     typedef struct
00033     {
00034         double kS;
00035         double KV;
00036         double kA;
00037         double kG;
00038     } ff_config_t;
00039
00049
00054     FeedForward(ff_config_t &cfg) : cfg(cfg) {}
00055
00066     double calculate(double v, double a, double pid_ref=0.0)
00067     {
00068         double ks_sign = 0;
00069         if(v != 0)
00070             ks_sign = sign(v);
00071         else if(pid_ref != 0)
00072             ks_sign = sign(pid_ref);
00073
00074         return (cfg.kS * ks_sign) + (cfg.KV * v) + (cfg.kA * a) + cfg.kG;
00075     }
00076
00077     private:
00078     ff_config_t &cfg;
00079
00080 };
00081
00083
00091 FeedForward::ff_config_t tune_feedforward(vex::motor_group &motor, double pct, double duration);

```

6.24 motion_controller.h

```

00001 #pragma once
00002 #include "../core/include/utils/controls/pid.h"
00003 #include "../core/include/utils/controls/feedforward.h"
00004 #include "../core/include/utils/controls/trapezoid_profile.h"
00005 #include "../core/include/utils/controls/feedback_base.h"
00006 #include "../core/include/subsystems/tank_drive.h"
00007 #include "../core/include/subsystems/screen.h"
00008
00009 #include "vex.h"
00010
00027 class MotionController : public Feedback
00028 {
00029     public:
00030
00036     typedef struct
00037     {
00038         double max_v;
00039         double accel;
00040         PID::pid_config_t pid_cfg;
00041         FeedForward::ff_config_t ff_cfg;
00042     } m_profile_cfg_t;
00043
00053     MotionController(m_profile_cfg_t &config);
00054
00059     void init(double start_pt, double end_pt, double start_vel, double end_vel) override;
00060
00067     double update(double sensor_val) override;
00068
00072     double get() override;
00073
00081     void set_limits(double lower, double upper) override;
00082
00087     bool is_on_target() override;
00088
00092     motion_t get_motion() const;
00093
00094
00095     screen::Page *Page();
00096
00115     static FeedForward::ff_config_t tune_feedforward(TankDrive &drive, OdometryTank &odometry, double
00096     pct=0.6, double duration=2);

```

```
00116     private:
00117     m_profile_cfg_t config;
00118
00119     PID pid;
00120     FeedForward ff;
00121     TrapezoidProfile profile;
00122
00123     double current_pos;
00124     double end_pt;
00125
00126     double lower_limit = 0, upper_limit = 0;
00127     double out = 0;
00128     motion_t cur_motion;
00129
00130     vex::timer tmr;
00131     friend class MotionControllerPage;
00132
00133 };
00134
00135 };
```

6.25 pid.h

```
00001 #pragma once
00002
00003 #include "../core/include/utils/controls/feedback_base.h"
00004 #include "vex.h"
00005 #include <cmath>
00006
00007 using namespace vex;
00008
00009 class PID : public Feedback {
00010 public:
00011     enum ERROR_TYPE {
00012         LINEAR,
00013         ANGULAR // assumes degrees
00014     };
00015     struct pid_config_t {
00016         double p;
00017         double i;
00018         double d;
00019         double deadband;
00020         double on_target_time;
00021         ERROR_TYPE error_method;
00022     };
00023     PID(pid_config_t &config);
00024
00025     void init(double start_pt, double set_pt, double start_vel = 0,
00026               double end_vel = 0) override;
00027
00028     double update(double sensor_val) override;
00029
00030     double get_sensor_val() const;
00031
00032     double get() override;
00033
00034     void set_limits(double lower, double upper) override;
00035
00036     bool is_on_target() override;
00037
00038     void reset();
00039
00040     double get_error();
00041
00042     double get_target() const;
00043
00044     void set_target(double target);
00045
00046     pid_config_t
00047     &config;
00048
00049 private:
00050     double last_error =
00051         0;
00052     double accum_error =
00053         0;
00054
00055     double last_time = 0;
00056     double on_target_last_time =
00057         0;
00058     double lower_limit =
```

```

00152      0;
00153  double upper_limit =
00154  0;
00155
00156  double target = 0;
00158  double target_vel = 0;
00160  double sensor_val = 0;
00162  double out = 0;
00165
00166  bool is_checking_on_target =
00167  false;
00168
00169  timer pid_timer;
00172 };

```

6.26 pidff.h

```

00001 #pragma once
00002 #include "../core/include/utils/controls/feedback_base.h"
00003 #include "../core/include/utils/controls/feedforward.h"
00004 #include "../core/include/utils/controls/pid.h"
00005
00006 class PIDFF : public Feedback {
00007 public:
00008     PIDFF(PID::pid_config_t &pid_cfg, FeedForward::ff_config_t &ff_cfg);
00009
00018 void init(double start_pt, double set_pt, double start_vel,
00019             double end_vel) override;
00020
00025 void set_target(double set_pt);
00026
00027 double get_target() const;
00028 double get_sensor_val() const;
00036 double update(double val) override;
00037
00046 double update(double val, double vel_setpt, double a_setpt = 0);
00047
00051 double get() override;
00052
00060 void set_limits(double lower, double upper) override;
00061
00065 bool is_on_target() override;
00066
00067 void reset();
00068
00069 PID pid;
00070
00071 private:
00072     FeedForward::ff_config_t &ff_cfg;
00073
00074     FeedForward ff;
00075
00076     double out;
00077     double lower_lim, upper_lim;
00078 };

```

6.27 take_back_half.h

```

00001 #pragma once
00002 #include "../core/include/utils/controls/feedback_base.h"
00003
00006 class TakeBackHalf : public Feedback
00007 {
00008
00009 public:
0010     TakeBackHalf(double TBH_gain, double first_cross_split, double on_target_threshold);
0019     void init(double start_pt, double set_pt, double, double);
0026     double update(double val) override;
0027
0031     double get() override;
0032
0039     void set_limits(double lower, double upper) override;
0040
0044     bool is_on_target() override;
0045
0046     double TBH_gain;
0047     double first_cross_split;
0048 private:
0049     double on_target_threshold;

```

```

00050     double target = 0.0;
00051
00052     bool first_cross = true;
00053     double tbh = 0.0;
00054     double prev_error = 0.0;
00055
00056     double output = 0.0;
00057     double lower = 0.0, upper = 0.0;
00058 }
00059 
```

6.28 trapezoid_profile.h

```

00001 #pragma once
00002
00003 const int MAX_TRAPEZOID_PROFILE_SEGMENTS = 4;
00004
00005 typedef struct {
00006     double pos;
00007     double vel;
00008     double accel;
00009 }
00010 } motion_t;
00011
00012 typedef struct {
00013     double pos_after;
00014     double vel_after;
00015     double accel;
00016     double duration;
00017 } trapezoid_profile_segment_t;
00018
00019 class TrapezoidProfile {
00020     public:
00021     TrapezoidProfile(double max_v, double accel);
00022
00023     motion_t calculate(double time_s, double pos_s);
00024
00025     motion_t calculate_time_based(double time_s);
00026
00027     void set_endpts(double start, double end);
00028
00029     void set_vel_endpts(double start, double end);
00030
00031     void set_accel(double accel);
00032
00033     void set_max_v(double max_v);
00034
00035     double get_movement_time() const;
00036
00037     double get_max_v() const;
00038     double get_accel() const;
00039
00040     private:
00041     double si, sf;
00042     double vi, vf;
00043     double max_v;
00044     double accel;
00045     double duration;
00046
00047     trapezoid_profile_segment_t segments[MAX_TRAPEZOID_PROFILE_SEGMENTS];
00048     int num_acceleration_phases;
00049
00050     bool precalculated;
00051
00052     bool precalculate();
00053
00054     trapezoid_profile_segment_t calculate_kinetic_motion(double si, double vi,
00055                                         double v_target);
00056
00057     trapezoid_profile_segment_t calculate_next_segment(double s, double v);
00058 }
00059 
```

6.29 generic_auto.h

```

00001 #pragma once
00002
00003 #include <queue>
00004 #include <map>
00005 #include "vex.h"

```

```

00006 #include <functional>
00007
00008 typedef std::function<bool(void)> state_ptr;
00009
0014 class GenericAuto
0015 {
0016     public:
0017
0031     [[deprecated("Use CommandController instead.")]]
0032     bool run(bool blocking);
0033
0038     [[deprecated("Use CommandController instead.")]]
0039     void add(state_ptr new_state);
0040
0045     [[deprecated("Use CommandController instead.")]]
0046     void add_async(state_ptr async_state);
0047
0052     [[deprecated("Use CommandController instead.")]]
0053     void add_delay(int ms);
0054
0055     private:
0056
0057     std::queue<state_ptr> state_list;
0058
0059 };

```

6.30 geometry.h

```

00001 #pragma once
00002 #include <cmath>
00003
00007 struct point_t
00008 {
00009     double x;
00010     double y;
00011
00017     double dist(const point_t other) const
00018     {
00019         return std::sqrt(std::pow(this->x - other.x, 2) + pow(this->y - other.y, 2));
00020     }
00021
00027     point_t operator+(const point_t &other) const
00028     {
00029         point_t p{
00030             .x = this->x + other.x,
00031             .y = this->y + other.y};
00032         return p;
00033     }
00034
00040     point_t operator-(const point_t &other) const
00041     {
00042         point_t p{
00043             .x = this->x - other.x,
00044             .y = this->y - other.y};
00045         return p;
00046     }
00047
00048     point_t operator*(double s) const
00049     {
00050         return {x * s, y * s};
00051     }
00052     point_t operator/(double s) const
00053     {
00054         return {x / s, y / s};
00055     }
00056
00057     point_t operator-() const
00058     {
00059         return {-x, -y};
00060     }
00061     point_t operator+() const
00062     {
00063         return {x, y};
00064     }
00065
00066     bool operator==(const point_t &rhs)
00067     {
00068         return x == rhs.x && y == rhs.y;
00069     }
00070 };
00071
00075 struct pose_t
00076 {

```

```

00077     double x;
00078     double y;
00079     double rot;
00080
00081     point_t get_point()
00082     {
00083         return point_t{.x = x, .y = y};
00084     }
00085
00086 } ;
00087
00088 struct Rect
00089 {
00090     point_t min;
00091     point_t max;
00092     static Rect from_min_and_size(point_t min, point_t size){
00093         return {min, min+size};
00094     }
00095     point_t dimensions() const
00096     {
00097         return max - min;
00098     }
00099     point_t center() const{
00100         return (min + max)/2;
00101     }
00102     double width() const{
00103         return max.x - min.x;
00104     }
00105     double height() const{
00106         return max.y - min.y;
00107     }
00108     bool contains(point_t p) const
00109     {
00110         bool xin = p.x > min.x && p.x < max.x;
00111         bool yin = p.y > min.y && p.y < max.y;
00112         return xin && yin;
00113     }
00114 }
00115 };
00116
00117 struct Mat2
00118 {
00119     double X11, X12;
00120     double X21, X22;
00121     point_t operator*(const point_t p) const
00122     {
00123         double outx = p.x * X11 + p.y * X12;
00124         double outy = p.x * X21 + p.y * X22;
00125         return {outx, outy};
00126     }
00127
00128     static Mat2 FromRotationDegrees(double degrees)
00129     {
00130         double rad = degrees * (M_PI / 180.0);
00131         double c = cos(rad);
00132         double s = sin(rad);
00133         return {c, -s, s, c};
00134     }
00135 };

```

6.31 graph_drawer.h

```

00001 #pragma once
00002
00003 #include <string>
00004 #include <stdio.h>
00005 #include <vector>
00006 #include <cmath>
00007 #include "vex.h"
00008 #include "../core/include/utils/geometry.h"
00009 #include "../core/include/utils/vector2d.h"
00010
00011 class GraphDrawer
00012 {
00013 public:
00020     GraphDrawer(int num_samples, double lower_bound, double upper_bound, std::vector<vex::color> colors,
00021     size_t num_series = 1);
00025     void add_samples(std::vector<point_t> sample);
00026
00031     void add_samples(std::vector<double> sample);
00032
00040     void draw(vex::brain::lcd &screen, int x, int y, int width, int height);
00041

```

```

00042 private:
00043     std::vector<std::vector<point_t>> series;
00044     int sample_index = 0;
00045     std::vector<vex::color> cols;
00046     vex::color bgcol = vex::transparent;
00047     bool border;
00048     double upper;
00049     double lower;
00050     bool auto_fit = false;
00051 };

```

6.32 logger.h

```

00001 #pragma once
00002
00003 #include <cstdarg>
00004 #include <cstdio>
00005 #include <string>
00006 #include "vex.h"
00007
00009 enum LogLevel
00010 {
00011     DEBUG,
00012     NOTICE,
00013     WARNING,
00014     ERROR,
00015     CRITICAL,
00016     TIME
00017 };
00018
00019 class Logger
00020 {
00021     private:
00022         const std::string filename;
00023         vex::brain::sdcard sd;
00024         void write_level(LogLevel l);
00025
00026     public:
00027         static constexpr int MAX_FORMAT_LEN = 512;
00028         explicit Logger(const std::string &filename);
00029
00030         Logger(const Logger &l) = delete;
00031         Logger &operator=(const Logger &l) = delete;
00032
00033
00034
00035         void Log(const std::string &s);
00036
00037         void Log(LogLevel level, const std::string &s);
00038
00039         void Logln(const std::string &s);
00040
00041         void Logln(LogLevel level, const std::string &s);
00042
00043         void Logf(const char *fmt, ...);
00044
00045         void Logf(LogLevel level, const char *fmt, ...);
00046
00047     };
00048 };

```

6.33 math_util.h

```

00001 #pragma once
00002 #include <vector>
00003 #include "math.h"
00004 #include "vex.h"
00005 #include "../core/include/utils/geometry.h"
00006
00007
00015 double clamp(double value, double low, double high);
00016
00023 double lerp(double a, double b, double t);
00030 double sign(double x);
00031
00032 double wrap_angle_deg(double input);
00033 double wrap_angle_rad(double input);
00034
00035 /**
00036 Calculates the variance of a set of numbers (needed for linear regression)
00037 https://en.wikipedia.org/wiki/Variance
00038 @param values the values for which the variance is taken

```

```

00039 @param mean      the average of values
00040 */
00041 double variance(std::vector<double> const &values, double mean);
00042
00043
00044 /*
00045 Calculates the average of a vector of doubles
00046 @param values    the list of values for which the average is taken
00047 */
00048 double mean(std::vector<double> const &values);
00049
00050 /*
00051 Calculates the covariance of a set of points (needed for linear regression)
00052 https://en.wikipedia.org/wiki/Covariance
00053
00054 @param points    the points for which the covariance is taken
00055 @param meanx     the mean value of all x coordinates in points
00056 @param meany     the mean value of all y coordinates in points
00057 */
00058 double covariance(std::vector<std::pair<double, double> const &points, double meanx, double meany);
00059
00060 /*
00061 Calculates the slope and y intercept of the line of best fit for the data
00062 @param points the points for the data
00063 */
00064 std::pair<double, double> calculate_linear_regression(std::vector<std::pair<double, double> const &points);
00065
00066 double estimate_path_length(const std::vector<point_t> &points);

```

6.34 moving_average.h

```

00001 #pragma once
00002 #include <vector>
00003
00008 class Filter
00009 {
00010 public:
00011     virtual void add_entry(double n) = 0;
00012     virtual double get_value() const = 0;
00013 };
00014
00027 class MovingAverage : public Filter
00028 {
00029 public:
00030     /*
00031     * Create a moving average calculator with 0 as the default value
00032     *
00033     * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
00034     reading
00035     */
00036     MovingAverage(int buffer_size);
00037     /*
00038     * Create a moving average calculator with a specified default value
00039     * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
00040     reading
00041     * @param starting_value The value that the average will be before any data is added
00042     */
00043     /*
00044     * Add a reading to the buffer
00045     * Before:
00046     * [ 1 1 2 2 3 3 ] => 2
00047     * ^
00048     * After:
00049     * [ 2 1 2 2 3 3 ] => 2.16
00050     * ^
00051     * @param n   the sample that will be added to the moving average.
00052     */
00053     void add_entry(double n) override;
00054
00055     double get_value() const override;
00060
00065     int get_size() const;
00066
00067 private:
00068     int buffer_index;           // index of the next value to be overridden
00069     std::vector<double> buffer; // all current data readings we've taken
00070     double current_avg;        // the current value of the data
00071 };
00072
00085 class ExponentialMovingAverage : public Filter

```

```

00086 {
00087     public:
00088     /*
00089     * Create a moving average calculator with 0 as the default value
00090     *
00091     * @param buffer_size      The size of the buffer. The number of samples that constitute a valid
00092     * reading
00093     */
00094     ExponentialMovingAverage(int buffer_size);
00095     /*
00096     * Create a moving average calculator with a specified default value
00097     * @param buffer_size      The size of the buffer. The number of samples that constitute a valid
00098     * reading
00099     * @param starting_value   The value that the average will be before any data is added
00100    */
00100    ExponentialMovingAverage(int buffer_size, double starting_value);
00101    /*
00102     * Add a reading to the buffer
00103     * Before:
00104     * [ 1 1 2 2 3 3 ] => 2
00105     *      ^
00106     * After:
00107     * [ 2 1 2 2 3 3 ] => 2.16
00108     *      ^
00109     * @param n    the sample that will be added to the moving average.
00110    */
00111    void add_entry(double n) override;
00112
00113    double get_value() const override;
00114
00115    int get_size();
00116
00117 private:
00118    int buffer_index;           // index of the next value to be overridden
00119    std::vector<double> buffer; // all current data readings we've taken
00120    double current_avg;        // the current value of the data
00121 };

```

6.35 pure_pursuit.h

```

00001 #pragma once
00002
00003 #include <vector>
00004 #include "../core/include/utils/geometry.h"
00005 #include "../core/include/utils/vector2d.h"
00006 #include "vex.h"
00007
00008 using namespace vex;
00009
00010 namespace PurePursuit {
00011     class Path
00012     {
00013     public:
00014         Path(std::vector<point_t> points, double radius);
00015         std::vector<point_t> get_points();
00016         double get_radius();
00017         bool is_valid();
00018
00019     private:
00020         std::vector<point_t> points;
00021         double radius;
00022         bool valid;
00023     };
00024     struct spline
00025     {
00026         double a, b, c, d, x_start, x_end;
00027
00028         double getY(double x)
00029         {
00030             return a * pow((x - x_start), 3) + b * pow((x - x_start), 2) + c * (x - x_start) + d;
00031         }
00032     };
00033     struct hermite_point
00034     {
00035         double x;
00036         double y;
00037         double dir;
00038         double mag;
00039
00040         point_t getPoint() const {

```

```

00068     return {x, y};
00069 }
00070
00071     Vector2D getTangent() const {
00072         return Vector2D(dir, mag);
00073     }
00074 };
00075
00076     extern std::vector<point_t> line_circle_intersections(point_t center, double r, point_t point1,
00077     point_t point2);
00078     extern point_t get_looking_ahead(const std::vector<point_t> &path, pose_t robot_loc, double radius);
00079
00080     extern std::vector<point_t> inject_path(const std::vector<point_t> &path, double spacing);
00081
00082     extern std::vector<point_t> smooth_path(const std::vector<point_t> &path, double weight_data, double
00083     weight_smooth, double tolerance);
00084
00085     extern std::vector<point_t> smooth_path_cubic(const std::vector<point_t> &path, double res);
00086
00087     extern std::vector<point_t> smooth_path_hermite(const std::vector<hermite_point> &path, double
00088     step);
00089
00090     extern double estimate_remaining_dist(const std::vector<point_t> &path, pose_t robot_pose, double
00091     radius);
00092
00093 }
```

6.36 serializer.h

```

00001 #pragma once
00002 #include <algorithm>
00003 #include <map>
00004 #include <string>
00005 #include <vector>
00006 #include <stdio.h>
00007 #include <vex.h>
00008
00009 const char serialization_separator = '$';
00010 const std::size_t MAX_FILE_SIZE = 4096;
00011
00012 class Serializer
00013 {
00014     private:
00015         bool flush_always;
00016         std::string filename;
00017         std::map<std::string, int> ints;
00018         std::map<std::string, bool> bools;
00019         std::map<std::string, double> doubles;
00020         std::map<std::string, std::string> strings;
00021
00022     public:
00023         ~Serializer()
00024         {
00025             save_to_disk();
00026             printf("Saving %s\n", filename.c_str());
00027             fflush(stdout);
00028         }
00029
00030         explicit Serializer(const std::string &filename, bool flush_always = true) :
00031             flush_always(flush_always), filename(filename), ints({}), bools({}), doubles({}),
00032             strings({})
00033     {
00034         read_from_disk();
00035     }
00036
00037         void save_to_disk() const;
00038
00039         void set_int(const std::string &name, int i);
00040         void set_bool(const std::string &name, bool b);
00041         void set_double(const std::string &name, double d);
00042         void set_string(const std::string &name, std::string str);
00043
00044         int int_or(const std::string &name, int otherwise);
00045         bool bool_or(const std::string &name, bool otherwise);
00046
00047 }
```

```
00090     double double_or(const std::string &name, double otherwise);
00091
00096     std::string string_or(const std::string &name, std::string otherwise);
00097 };
```

6.37 state_machine.h

```
00001 #pragma once
00002 #include <string>
00003 #include <type_traits>
00004 #include <utility>
00005
00034 template <typename System, typename IDType, typename Message, int32_t delay_ms,
00035         bool do_log = false>
00036 class StateMachine {
00037     static_assert(std::is_enum<Message>::value,
00038                 "Message should be an enum (it's easier that way)");
00039     static_assert(std::is_enum<IDType>::value,
00040                 "IDType should be an enum (it's easier that way)");
00041
00042     public:
00043         class MaybeMessage {
00044             public:
00045                 MaybeMessage() : exists(false) {}
00046                 MaybeMessage(Message msg) : exists(true), thing(msg) {}
00047                 bool has_message() { return exists; }
00048                 Message message() { return thing; }
00049
00050             private:
00051                 bool exists;
00052                 Message thing;
00053         };
00054         struct State {
00055             // run once when we enter the state
00056             virtual void entry(System &) {}
00057             // run continuously while in the state
00058             virtual MaybeMessage work(System &) { return {}; }
00059             // run once when we exit the state
00060             virtual void exit(System &) {}
00061             // respond to a message when one comes in
00062             virtual State *respond(System &s, Message m) = 0;
00063             // Identify
00064             virtual IDType id() const = 0;
00065
00066             // virtual destructor cuz c++
00067             virtual ~State() {}
00068         };
00069
00070         // Data that gets passed to the runner thread. Don't worry too much about
00071         // this
00072         using thread_data = std::pair<State *, StateMachine *>;
00073
00074     StateMachine(State *initial)
00075         : runner(thread_runner, new thread_data{initial, this}) {}
00076
00077     IDType current_state() const {
00078         mut.lock();
00079         auto t = cur_type;
00080         mut.unlock();
00081         return t;
00082     }
00083     void send_message(Message msg) {
00084         mut.lock();
00085         incoming_msg = msg;
00086         mut.unlock();
00087     }
00088
00089     private:
00090         vex::task runner;
00091         mutable vex::mutex mut;
00092         MaybeMessage incoming_msg;
00093         IDType cur_type;
00094
00095         static int thread_runner(void *vptr) {
00096             thread_data *ptr = static_cast<thread_data *>(vptr);
00097             State *cur_state = ptr->first;
00098
00099             StateMachine &sys = *ptr->second;
00100             System &derived = *static_cast<System *>(&sys);
00101
00102             cur_state->entry(derived);
00103             sys.cur_type = cur_state->id();
00104         }
00105     };
00106 }
```

```

00152     auto respond_to_message = [&] (Message msg) {
00153         if (do_log) {
00154             printf("responding to msg: %s\n", to_string(msg).c_str());
00155             fflush(stdout);
00156         }
00157
00158         State *next_state = cur_state->respond(derived, msg);
00159
00160         if (cur_state != next_state) {
00161             // switched states
00162             sys.mut.lock();
00163
00164             cur_state->exit(derived);
00165             next_state->entry(derived);
00166
00167             delete cur_state;
00168
00169             cur_state = next_state;
00170             sys.cur_type = cur_state->id();
00171
00172             sys.mut.unlock();
00173         }
00174     };
00175
00176     while (true) {
00177         if (do_log) {
00178             std::string str = to_string(cur_state->id());
00179             std::string str2 = to_string(sys.cur_type);
00180
00181             printf("state: %s %s\n", str.c_str(), str2.c_str());
00182         }
00183
00184         // Internal Message passed
00185         MaybeMessage internal_msg = cur_state->work(derived);
00186
00187         if (internal_msg.has_message()) {
00188             respond_to_message(internal_msg.message());
00189         }
00190
00191         // External Message passed
00192         sys.mut.lock();
00193         MaybeMessage incoming = sys.incoming_msg;
00194         sys.incoming_msg = {};
00195         sys.mut.unlock();
00196
00197         if (incoming.has_message()) {
00198             respond_to_message(incoming.message());
00199         }
00200
00201         vexDelay(delay_ms);
00202     }
00203
00204     return 0;
00205 }
00206 };

```

6.38 vector2d.h

```

00001 #pragma once
00002
00003
00004 #include <cmath>
00005 #include "../core/include/utils/geometry.h"
00006
00007 #ifndef PI
00008 #define PI 3.141592654
00009 #endif
00010 class Vector2D
00011 {
00012     public:
00013         Vector2D(double dir, double mag);
00014
00015         Vector2D(point_t p);
00016
00017         double get_dir() const;
00018
00019         double get_mag() const;
00020
00021         double get_x() const;
00022
00023         double get_y() const;
00024
00025         Vector2D normalize();

```

```
00062     point_t point();
00063
00064     Vector2D operator*(const double &x);
00065     Vector2D operator+(const Vector2D &other);
00066     Vector2D operator-(const Vector2D &other);
00067
00068     private:
00069     double dir, mag;
00070
00071 };
00072
00073 double deg2rad(double deg);
00074
00075 double rad2deg(double r);
```

Index

accel
 OdometryBase, 102

add
 CommandController, 31, 32
 GenericAuto, 66

add_async
 GenericAuto, 66

add_cancel_func
 CommandController, 32

add_delay
 CommandController, 33
 GenericAuto, 66

add_entry
 ExponentialMovingAverage, 46
 Filter, 53
 MovingAverage, 92

add_samples
 GraphDrawer, 68

AndCondition, 13
 test, 13

ang_accel_deg
 OdometryBase, 102

ang_speed_deg
 OdometryBase, 102

Async, 14
 run, 15

auto_drive
 MecanumDrive, 83

auto_turn
 MecanumDrive, 83

AutoChooser, 15
 AutoChooser, 16
 choice, 17
 draw, 16
 get_choice, 16
 list, 17
 update, 17

AutoChooser::entry_t, 44
 name, 44

AutoCommand, 18
 on_timeout, 19
 run, 19
 timeout_seconds, 19

background_task
 OdometryBase, 99

BangBang, 20
 get, 20
 init, 20
 is_on_target, 21

set_limits, 21
 update, 21

BasicSolenoidSet, 22
 BasicSolenoidSet, 22
 run, 23

BasicSpinCommand, 23
 BasicSpinCommand, 24
 run, 25

BasicStopCommand, 25
 BasicStopCommand, 26
 run, 26

bool_or
 Serializer, 137

BrakeType
 TankDrive, 152

Branch, 27
 on_timeout, 28
 run, 28

ButtonWidget
 screen::ButtonWidget, 29

calculate
 FeedForward, 50
 TrapezoidProfile, 165

calculate_time_based
 TrapezoidProfile, 165

choice
 AutoChooser, 17

CommandController, 30
 add, 31, 32
 add_cancel_func, 32
 add_delay, 33
 CommandController, 31
 last_command_timed_out, 33
 run, 33

Condition, 33

config
 PID, 120

control_continuous
 Lift< T >, 73

control_manual
 Lift< T >, 73

control_setpoints
 Lift< T >, 74

Core, 1

current_pos
 OdometryBase, 102

current_state
 StateMachine< System, IDType, Message, delay_ms, do_log >, 146

CustomEncoder, 34
 CustomEncoder, 34
 position, 35
 rotation, 35
 setPosition, 35
 setRotation, 36
 velocity, 36

DelayCommand, 36
 DelayCommand, 37
 run, 37

dist
 point_t, 128

double_or
 Serializer, 137

draw
 AutoChooser, 16
 FlywheelPage, 57
 GraphDrawer, 68
 MotionControllerPage, 90
 screen::FunctionPage, 65
 screen::OdometryPage, 104
 screen::Page, 111
 screen::PIDPage, 126
 screen::StatsPage, 147
 screen::WidgetPage, 179
 VideoPlayer, 174

drive
 MecanumDrive, 84

drive_ arcade
 TankDrive, 152

drive_ forward
 TankDrive, 153

drive_ raw
 MecanumDrive, 84

drive_ tank
 TankDrive, 154

drive_ tank_ raw
 TankDrive, 155

drive_ to_ point
 TankDrive, 155, 156

DriveForwardCommand, 38
 DriveForwardCommand, 39
 on_timeout, 39
 run, 40

DriveStopCommand, 40
 DriveStopCommand, 41
 on_timeout, 41
 run, 41

DriveToPointCommand, 42
 DriveToPointCommand, 43
 run, 44

end_async
 OdometryBase, 99

error_method
 PID::pid_config_t, 120

ERROR_TYPE
 PID, 116

ExponentialMovingAverage, 45
 add_entry, 46
 ExponentialMovingAverage, 45
 get_size, 46
 get_value, 46

Feedback, 47
 get, 48
 init, 48
 is_on_target, 48
 set_limits, 48
 update, 49

FeedForward, 49
 calculate, 50
 FeedForward, 50

FeedForward::ff_config_t, 51
 kA, 51
 kG, 51
 kS, 52
 kV, 52

Filter, 52
 add_entry, 53
 get_value, 53

Flywheel, 53
 Flywheel, 54
 get_motors, 54
 get_target, 54
 getRPM, 54
 is_on_target, 55
 Page, 55
 spin_manual, 55
 spin_rpm, 56
 SpinRpmCmd, 56
 spinRPMTask, 57
 stop, 56
 WaitUntilUpToSpeedCmd, 56

FlywheelPage, 57
 draw, 57
 update, 57

FlywheelStopCommand, 58
 FlywheelStopCommand, 59
 run, 59

FlywheelStopMotorsCommand, 59
 FlywheelStopMotorsCommand, 60
 run, 60

FlywheelStopNonTasksCommand, 61

FunctionCommand, 62
 run, 63

FunctionCondition, 63
 test, 64

FunctionPage
 screen::FunctionPage, 64

GenericAuto, 65
 add, 66
 add_async, 66
 add_delay, 66
 run, 67

get

BangBang, 20
Feedback, 48
MotionController, 88
PID, 117
PIDFF, 121
TakeBackHalf, 149
get_accel
 OdometryBase, 99
get_angular_accel_deg
 OdometryBase, 99
get_angular_speed_deg
 OdometryBase, 99
get_async
 Lift< T >, 74
get_choice
 AutoChooser, 16
get_dir
 Vector2D, 172
get_error
 PID, 117
get_mag
 Vector2D, 172
get_motion
 MotionController, 88
get_motors
 Flywheel, 54
get_movement_time
 TrapezoidProfile, 165
get_points
 PurePursuit::Path, 115
get_position
 OdometryBase, 100
get_radius
 PurePursuit::Path, 115
get_sensor_val
 PID, 117
get_setpoint
 Lift< T >, 74
get_size
 ExponentialMovingAverage, 46
 MovingAverage, 93
get_speed
 OdometryBase, 100
get_target
 Flywheel, 54
 PID, 117
get_value
 ExponentialMovingAverage, 46
 Filter, 53
 MovingAverage, 93
get_x
 Vector2D, 172
get_y
 Vector2D, 172
getRPM
 Flywheel, 54
GraphDrawer, 67
 add_samples, 68
draw, 68
GraphDrawer, 67
handle
 OdometryBase, 103
has_message
 StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage, 81
hold
 Lift< T >, 74
home
 Lift< T >, 74
IfTimePassed, 69
 test, 70
include/robot_specs.h, 181
include/subsystems/custom_encoder.h, 181
include/subsystems/flywheel.h, 182
include/subsystems/fun/pl_mpeg.h, 182
include/subsystems/fun/video.h, 232
include/subsystems/layout.h, 232
include/subsystems/lift.h, 232
include/subsystems/mecanum_drive.h, 235
include/subsystems/odometry/odometry_3wheel.h, 236
include/subsystems/odometry/odometry_base.h, 236
include/subsystems/odometry/odometry_tank.h, 237
include/subsystems/screen.h, 238
include/subsystems/tank_drive.h, 240
include/utils/auto_chooser.h, 242
include/utils/command_structure/auto_command.h, 242
include/utils/command_structure/basic_command.h,
 245
include/utils/command_structure/command_controller.h,
 245
include/utils/command_structure/delay_command.h,
 246
include/utils/command_structure/drive_commands.h,
 246
include/utils/command_structure/flywheel_commands.h,
 248
include/utils/controls/bang_bang.h, 249
include/utils/controls/feedback_base.h, 249
include/utils/controls/feedforward.h, 249
include/utils/controls/motion_controller.h, 250
include/utils/controls/pid.h, 251
include/utils/controls/pidff.h, 252
include/utils/controls/take_back_half.h, 252
include/utils/controls/trapezoid_profile.h, 253
include/utils/generic_auto.h, 253
include/utils/geometry.h, 254
include/utils/graph_drawer.h, 255
include/utils/logger.h, 256
include/utils/math_util.h, 256
include/utils/moving_average.h, 257
include/utils/pure_pursuit.h, 258
include/utils/serializer.h, 259
include/utils/state_machine.h, 260
include/utils/vector2d.h, 261
init

BangBang, 20
Feedback, 48
MotionController, 88
PID, 118
PIDFF, 121
TakeBackHalf, 149
InOrder, 70
on_timeout, 71
run, 71
int_or
 Serializer, 138
is_on_target
 BangBang, 21
 Feedback, 48
 Flywheel, 55
 MotionController, 88
 PID, 118
 PIDFF, 123
 TakeBackHalf, 149
is_valid
 PurePursuit::Path, 115
kA
 FeedForward::ff_config_t, 51
kG
 FeedForward::ff_config_t, 51
kS
 FeedForward::ff_config_t, 52
kV
 FeedForward::ff_config_t, 52
last_command_timed_out
 CommandController, 33
Lift
 Lift< T >, 73
Lift< T >, 72
 control_continuous, 73
 control_manual, 73
 control_setpoints, 74
 get_async, 74
 get_setpoint, 74
 hold, 74
 home, 74
 Lift, 73
 set_async, 75
 set_position, 75
 set_sensor_function, 75
 set_sensor_reset, 76
 set_setpoint, 76
Lift< T >::lift_cfg_t, 76
list
 AutoChooser, 17
Log
 Logger, 78
Logf
 Logger, 78
Logger, 77
 Log, 78
 Logf, 78
Logger, 77
 Logger, 77
 LogIn, 79
 Logger, 79
Mat2, 80
MaybeMessage
 StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage, 81
MecanumDrive, 82
 auto_drive, 83
 auto_turn, 83
 drive, 84
 drive_raw, 84
 MecanumDrive, 82
MecanumDrive::mecanumdrive_config_t, 85
message
 StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage, 81
modify_inputs
 TankDrive, 157
motion_t, 85
MotionController, 86
 get, 88
 get_motion, 88
 init, 88
 is_on_target, 88
 MotionController, 87
 set_limits, 88
 tune_feedforward, 89
 update, 89
MotionController::m_profile_cfg_t, 79
MotionControllerPage, 90
 draw, 90
 update, 91
MovingAverage, 91
 add_entry, 92
 get_size, 93
 get_value, 93
 MovingAverage, 92
mut
 OdometryBase, 103
name
 AutoChooser::entry_t, 44
None
 TankDrive, 152
normalize
 Vector2D, 172
Odometry3Wheel, 93
 Odometry3Wheel, 95
 tune, 95
 update, 96
Odometry3Wheel::odometry3wheel_cfg_t, 96
 off_axis_center_dist, 97
 wheel_diam, 97
 wheelbase_dist, 97
OdometryBase, 97

accel, 102
ang_accel_deg, 102
ang_speed_deg, 102
background_task, 99
current_pos, 102
end_async, 99
get_accel, 99
get-angular_accel_deg, 99
get-angular_speed_deg, 99
get_position, 100
get_speed, 100
handle, 103
mut, 103
OdometryBase, 98
pos_diff, 100
rot_diff, 101
set_position, 101
smallest_angle, 101
speed, 103
update, 102
zero_pos, 103
OdometryPage
 screen::OdometryPage, 104
OdometryTank, 105
 OdometryTank, 106, 107
 set_position, 108
 update, 108
OdomSetPosition, 108
 OdomSetPosition, 109
 run, 110
off_axis_center_dist
 Odometry3Wheel::odometry3wheel_cfg_t, 97
on_target_time
 PID::pid_config_t, 120
on_timeout
 AutoCommand, 19
 Branch, 28
 DriveForwardCommand, 39
 DriveStopCommand, 41
 InOrder, 71
 Parallel, 113
 PurePursuitCommand, 131
 RepeatUntil, 134
 TurnDegreesCommand, 168
 TurnToHeadingCommand, 170
operator+
 point_t, 128
 Vector2D, 173
operator-
 point_t, 129
 Vector2D, 173
operator*
 Vector2D, 173
OrCondition, 110
 test, 110
Page
 Flywheel, 55
Parallel, 112
on_timeout, 113
run, 113
parallel_runner_info, 114
Path
 PurePursuit::Path, 114
PID, 115
 config, 120
 ERROR_TYPE, 116
 get, 117
 get_error, 117
 get_sensor_val, 117
 get_target, 117
 init, 118
 is_on_target, 118
 PID, 116
 reset, 118
 set_limits, 118
 set_target, 119
 update, 119
PID::pid_config_t, 120
 error_method, 120
 on_target_time, 120
PIDFF, 121
 get, 121
 init, 121
 is_on_target, 123
 set_limits, 123
 set_target, 123
 update, 124
PIDPage
 screen::PIDPage, 125
plm_frame_t, 126
plm_packet_t, 127
plm_plane_t, 127
plm_samples_t, 127
point
 Vector2D, 174
point_t, 127
 dist, 128
 operator+, 128
 operator-, 129
pos_diff
 OdometryBase, 100
pose_t, 129
position
 CustomEncoder, 35
pure_pursuit
 TankDrive, 157, 158
PurePursuit::hermite_point, 69
PurePursuit::Path, 114
 get_points, 115
 get_radius, 115
 is_valid, 115
 Path, 114
PurePursuit::spline, 144
PurePursuitCommand, 130
 on_timeout, 131
 PurePursuitCommand, 131

run, 131
 Rect, 131
 RepeatUntil, 132
 on_timeout, 134
 RepeatUntil, 133, 134
 run, 134
 reset
 PID, 118
 reset_auto
 TankDrive, 159
 robot_specs_t, 135
 rot_diff
 OdometryBase, 101
 rotation
 CustomEncoder, 35
 run
 Async, 15
 AutoCommand, 19
 BasicSolenoidSet, 23
 BasicSpinCommand, 25
 BasicStopCommand, 26
 Branch, 28
 CommandController, 33
 DelayCommand, 37
 DriveForwardCommand, 40
 DriveStopCommand, 41
 DriveToPointCommand, 44
 FlywheelStopCommand, 59
 FlywheelStopMotorsCommand, 60
 FunctionCommand, 63
 GenericAuto, 67
 InOrder, 71
 OdomSetPosition, 110
 Parallel, 113
 PurePursuitCommand, 131
 RepeatUntil, 134
 SpinRPMCommand, 143
 TurnDegreesCommand, 168
 TurnToHeadingCommand, 170
 WaitUntilCondition, 176
 WaitUntilUpToSpeedCommand, 178

 save_to_disk
 Serializer, 138
 screen::ButtonConfig, 28
 screen::ButtonWidget, 28
 ButtonWidget, 29
 update, 30
 screen::CheckboxConfig, 30
 screen::FunctionPage, 64
 draw, 65
 FunctionPage, 64
 update, 65
 screen::LabelConfig, 72
 screen::OdometryPage, 103
 draw, 104
 OdometryPage, 104
 update, 104

 screen::Page, 111
 draw, 111
 update, 112
 screen::PIDPage, 125
 draw, 126
 PIDPage, 125
 update, 126
 screen::ScreenData, 135
 screen::ScreenRect, 136
 screen::SizedWidget, 140
 screen::SliderConfig, 140
 screen::SliderWidget, 141
 SliderWidget, 141
 update, 141
 screen::StatsPage, 147
 draw, 147
 StatsPage, 147
 update, 148
 screen::TextConfig, 162
 screen::WidgetConfig, 178
 screen::WidgetPage, 179
 draw, 179
 update, 179
 send_message
 StateMachine< System, IDType, Message, delay_ms, do_log >, 146
 Serializer, 136
 bool_or, 137
 double_or, 137
 int_or, 138
 save_to_disk, 138
 Serializer, 137
 set_bool, 138
 set_double, 139
 set_int, 139
 set_string, 139
 string_or, 139
 set_accel
 TrapezoidProfile, 166
 set_async
 Lift< T >, 75
 set_bool
 Serializer, 138
 set_double
 Serializer, 139
 set_endpts
 TrapezoidProfile, 166
 set_int
 Serializer, 139
 set_limits
 BangBang, 21
 Feedback, 48
 MotionController, 88
 PID, 118
 PIDFF, 123
 TakeBackHalf, 150
 set_max_v
 TrapezoidProfile, 166

set_position
 Lift< T >, 75
 OdometryBase, 101
 OdometryTank, 108
set_sensor_function
 Lift< T >, 75
set_sensor_reset
 Lift< T >, 76
set_setpoint
 Lift< T >, 76
set_string
 Serializer, 139
set_target
 PID, 119
 PIDFF, 123
set_vel_endpts
 TrapezoidProfile, 166
setPosition
 CustomEncoder, 35
setRotation
 CustomEncoder, 36
SliderCfg, 140
SliderWidget
 screen::SliderWidget, 141
smallest_angle
 OdometryBase, 101
Smart
 TankDrive, 152
speed
 OdometryBase, 103
spin_manual
 Flywheel, 55
spin_rpm
 Flywheel, 56
SpinRpmCmd
 Flywheel, 56
SpinRPMCommand, 142
 run, 143
 SpinRPMCommand, 143
spinRPTask
 Flywheel, 57
StateMachine
 StateMachine< System, IDType, Message, delay_ms, do_log >, 146
StateMachine< System, IDType, Message, delay_ms,
 do_log >, 145
 current_state, 146
 send_message, 146
 StateMachine, 146
StateMachine< System, IDType, Message, delay_ms,
 do_log >::MaybeMessage, 80
 has_message, 81
 MaybeMessage, 81
 message, 81
StateMachine< System, IDType, Message, delay_ms,
 do_log >::State, 144
StatsPage
 screen::StatsPage, 147
stop
 Flywheel, 56
 TankDrive, 159
string_or
 Serializer, 139
TakeBackHalf, 148
 get, 149
 init, 149
 is_on_target, 149
 set_limits, 150
 update, 150
TankDrive, 150
 BrakeType, 152
 drive_arcade, 152
 drive_forward, 153
 drive_tank, 154
 drive_tank_raw, 155
 drive_to_point, 155, 156
 modify_inputs, 157
 None, 152
 pure_pursuit, 157, 158
 reset_auto, 159
 Smart, 152
 stop, 159
 TankDrive, 152
 turn_degrees, 159
 turn_to_heading, 160, 161
 ZeroVelocity, 152
test
 AndCondition, 13
 FunctionCondition, 64
 IfTimePassed, 70
 OrCondition, 110
 TimesTestedCondition, 162
timeout_seconds
 AutoCommand, 19
TimesTestedCondition, 162
 test, 162
trapezoid_profile_segment_t, 162
TrapezoidProfile, 163
 calculate, 165
 calculate_time_based, 165
 get_movement_time, 165
 set_accel, 166
 set_endpts, 166
 set_max_v, 166
 set_vel_endpts, 166
 TrapezoidProfile, 164
tune
 Odometry3Wheel, 95
tune_feedforward
 MotionController, 89
turn_degrees
 TankDrive, 159
turn_to_heading
 TankDrive, 160, 161
TurnDegreesCommand, 167
 on_timeout, 168

run, 168
TurnDegreesCommand, 168
TurnToHeadingCommand, 169
on_timeout, 170
run, 170
TurnToHeadingCommand, 169

update
AutoChooser, 17
BangBang, 21
Feedback, 49
FlywheelPage, 57
MotionController, 89
MotionControllerPage, 91
Odometry3Wheel, 96
OdometryBase, 102
OdometryTank, 108
PID, 119
PIDFF, 124
screen::ButtonWidget, 30
screen::FunctionPage, 65
screen::OdometryPage, 104
screen::Page, 112
screen::PIDPage, 126
screen::SliderWidget, 141
screen::StatsPage, 148
screen::WidgetPage, 179
TakeBackHalf, 150
VideoPlayer, 175

Vector2D, 170
get_dir, 172
get_mag, 172
get_x, 172
get_y, 172
normalize, 172
operator+, 173
operator-, 173
operator*, 173
point, 174
Vector2D, 171

velocity
CustomEncoder, 36
VideoPlayer, 174
draw, 174
update, 175

WaitUntilCondition, 175
run, 176
WaitUntilUpToSpeedCmd
Flywheel, 56
WaitUntilUpToSpeedCommand, 176
run, 178
WaitUntilUpToSpeedCommand, 177

wheel_diam
Odometry3Wheel::odometry3wheel_cfg_t, 97
wheelbase_dist
Odometry3Wheel::odometry3wheel_cfg_t, 97