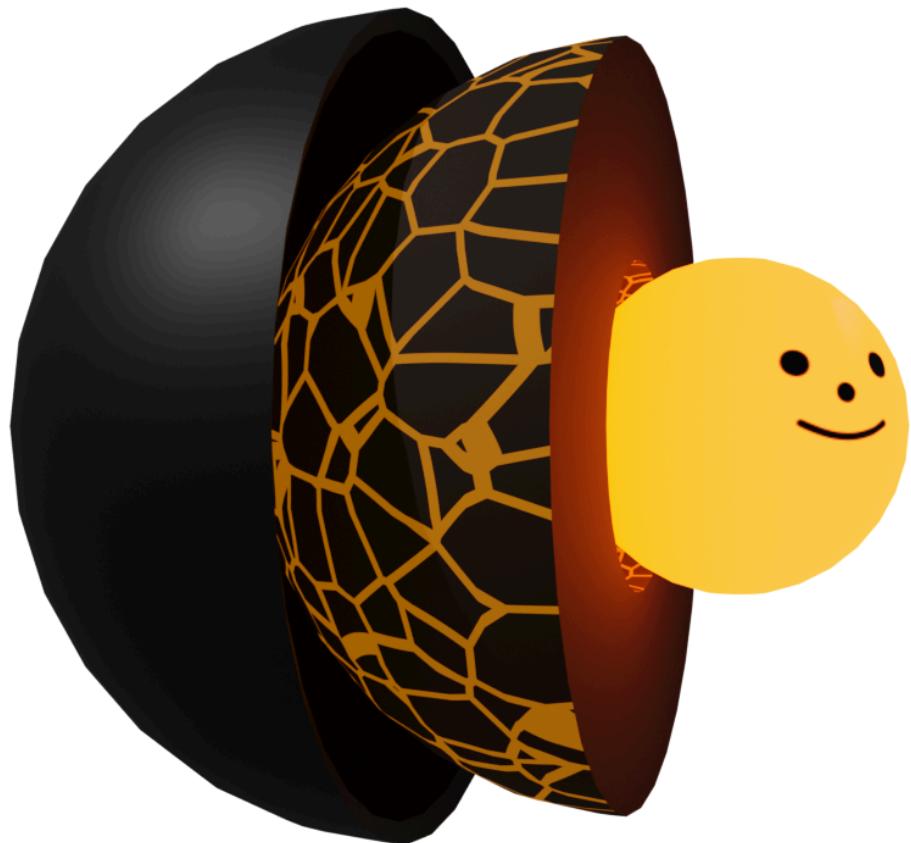


# RIT VEXU Software Engineering Notebook

2023-2024



# Table of Contents

<b>RIT VEXU Software Engineering Notebook.....</b>	<b>1</b>
<b>Table of Contents.....</b>	<b>2</b>
<b>Overview.....</b>	<b>3</b>
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
<b>Core: Fundamentals.....</b>	<b>7</b>
Odometry.....	7
Drivetrain.....	9
Tank Drivetrain.....	9
Control Loops.....	14
Auto Command Structure (ACS).....	16
Serializer.....	18
Screen Subsystem.....	19
Catapult System.....	22
<b>Core: Ongoing Projects.....</b>	<b>23</b>
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](https://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](https://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
Todo (20) This item hasn't been started	In Progress (4) This is actively being worked on	Done (13) This has been completed
Core #32 New Project Streamlining Wiki RIT-VEX-U/Core	Core #5 Pure Pursuit functionality RIT-VEX-U/Core	Core #44 ACS Command Timeout RIT-VEX-U/Core
Core #33 Core Cleanup / Documentation RIT-VEX-U/Core	Core #34 Motion Profiles RIT-VEX-U/Core	Core #39 Add 3 Pod Odometry to Core RIT-VEX-U/Core
Core #36 Wiki Entries RIT-VEX-U/Core	Core #73 Accelerometer for Lateral Odometry Tracking RIT-VEX-U/Core	Core #43 Add Pose2D Class RIT-VEX-U/Core

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.

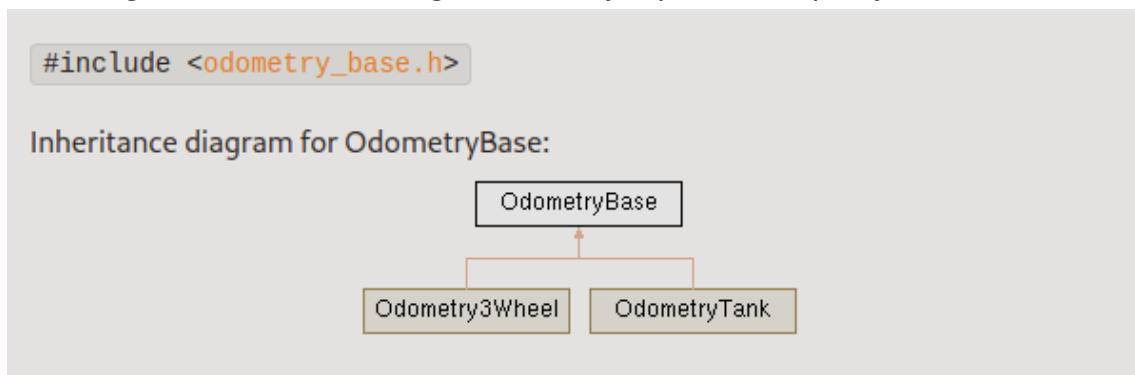


Figure 5: Automatically generated documentation.

## 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 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

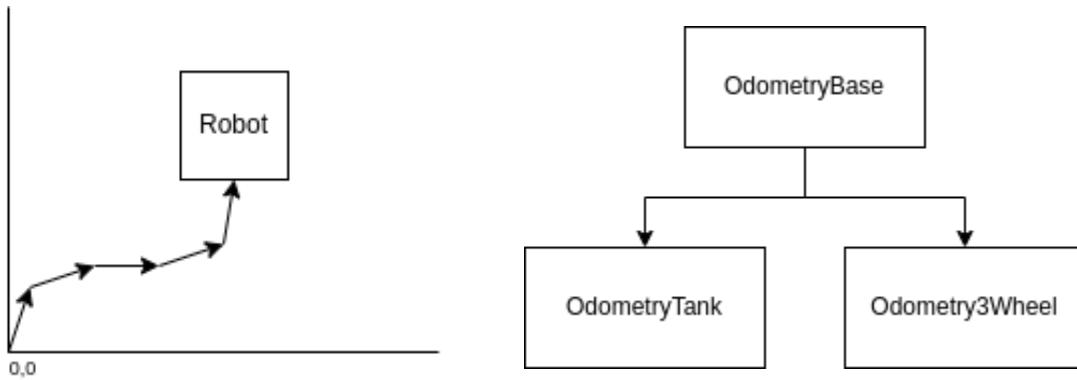
```
drive_system.drive_tank(controller.Axis3.position() / 100.0, controller.Axis2.position() / 100.0);
drive_system.drive_arcade(controller.Axis3.position() / 100.0, controller.Axis1.position() / 100.0);
```

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* ( $\alpha$ ). 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 ( $x, y$ ),  $\theta$  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  $2\pi$  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.

## 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.

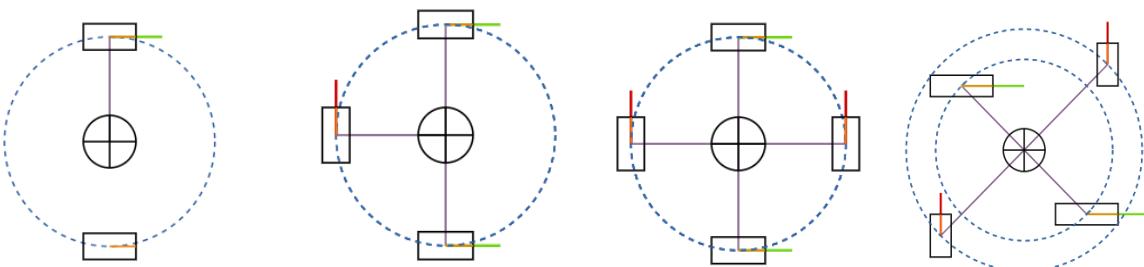


Figure 19: 2 pod, 3 pod, and arbitrary pods such a system could handle.

### 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  $E_1, E_2, \dots, E_3$  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 ( $\theta$ ), and a radius of the wheel ( $r$ ).

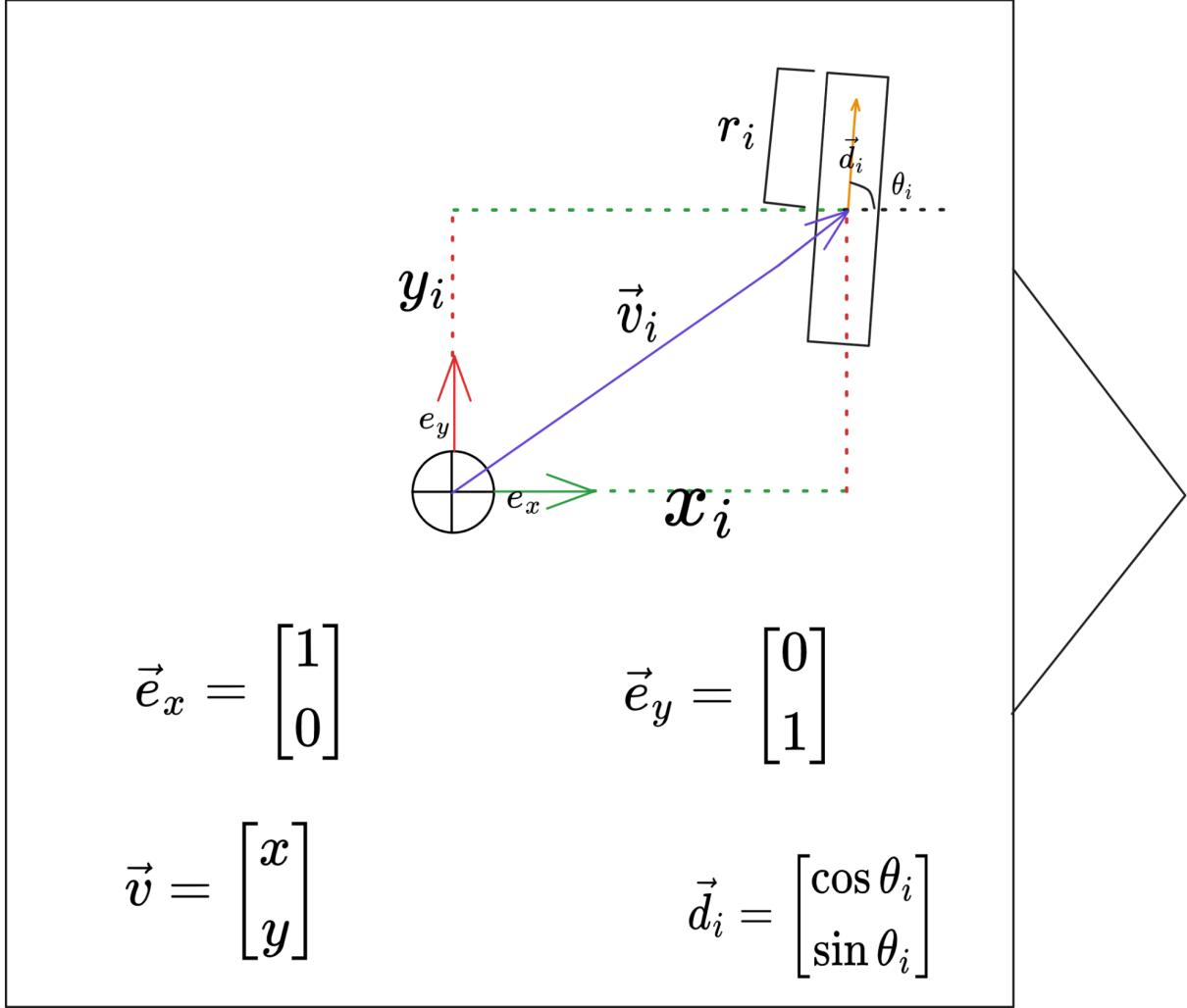


Figure 20: The configuration of a tracking wheel on the robot.  $(e_x, e_y)$  are the basis vectors of our coordinate system - the X and Y axes of the robot coordinate frame.  $d_i$  is the direction vector of the tracking wheel.

Now, if we pretend that these wheels are powered and we wish to translate and rotate the robot according to some controller input  $(x, y, \theta)$  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  $(x, y, \theta)$  (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  $\theta$  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,

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

$F_\theta$

$F_\theta$  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  $\theta_r$  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  $\vec{v}$ .

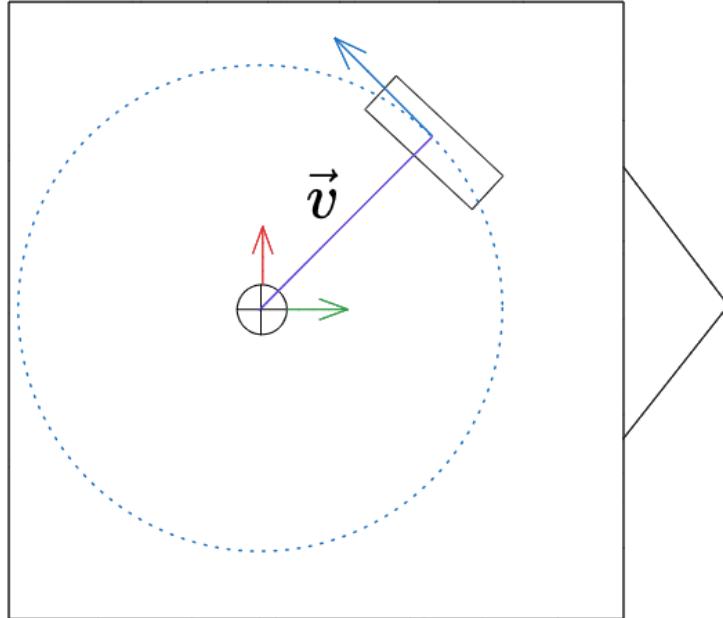
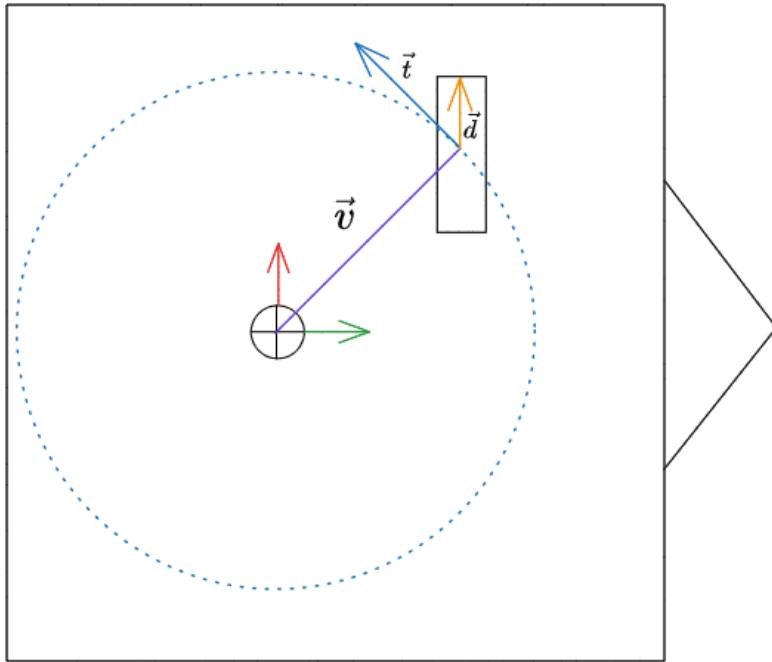


Figure 21: A conceptual perfectly perpendicular wheel

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

$$E_t r = S = \|\vec{v}\| \theta_r$$

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

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

So

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

## 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  $(x, y, \theta)$  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  $(x, y, \theta)$  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.

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

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

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

pose velocity

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

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.

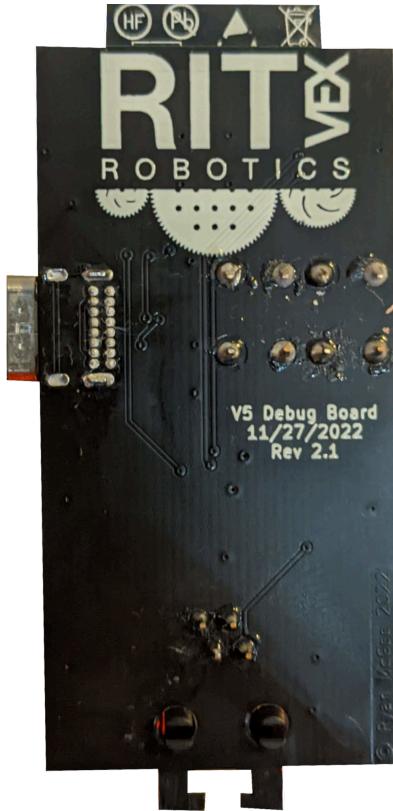


Figure 22 - Debug Board (Back)



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.

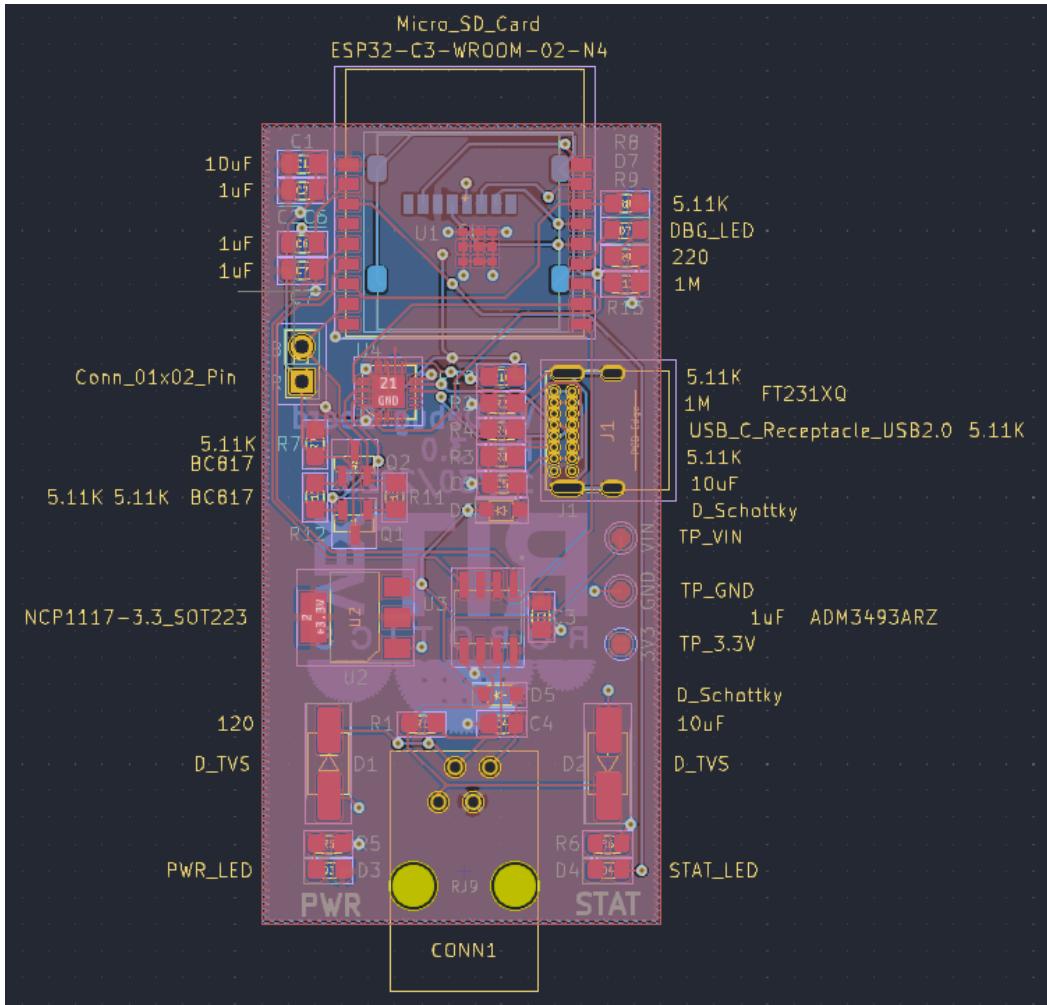


Figure 24 - Debug Board PCB Layout

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](https://github.com/superrm11/VexDebugBoard).

# RIT VEXU Core API

Generated by Doxygen 1.10.0



---

<b>1 Core</b>	<b>1</b>
1.1 Getting Started . . . . .	1
1.2 Features . . . . .	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy . . . . .	3
<b>3 Class Index</b>	<b>7</b>
3.1 Class List . . . . .	7
<b>4 File Index</b>	<b>11</b>
4.1 File List . . . . .	11
<b>5 Class Documentation</b>	<b>13</b>
5.1 AndCondition Class Reference . . . . .	13
5.1.1 Member Function Documentation . . . . .	14
5.1.1.1 test() . . . . .	14
5.2 Async Class Reference . . . . .	14
5.2.1 Detailed Description . . . . .	15
5.2.2 Member Function Documentation . . . . .	16
5.2.2.1 run() . . . . .	16
5.3 AutoChooser Class Reference . . . . .	16
5.3.1 Detailed Description . . . . .	17
5.3.2 Constructor & Destructor Documentation . . . . .	17
5.3.2.1 AutoChooser() . . . . .	17
5.3.3 Member Function Documentation . . . . .	18
5.3.3.1 draw() . . . . .	18
5.3.3.2 get_choice() . . . . .	18
5.3.3.3 update() . . . . .	18
5.3.4 Member Data Documentation . . . . .	19
5.3.4.1 choice . . . . .	19
5.3.4.2 list . . . . .	19
5.4 AutoCommand Class Reference . . . . .	20
5.4.1 Member Function Documentation . . . . .	21
5.4.1.1 on_timeout() . . . . .	21
5.4.1.2 run() . . . . .	22
5.4.2 Member Data Documentation . . . . .	22
5.4.2.1 timeout_seconds . . . . .	22
5.5 BangBang Class Reference . . . . .	22
5.5.1 Member Function Documentation . . . . .	23
5.5.1.1 get() . . . . .	23
5.5.1.2 init() . . . . .	23
5.5.1.3 is_on_target() . . . . .	24
5.5.1.4 set_limits() . . . . .	24

---

5.5.1.5 update()	24
5.6 BasicSolenoidSet Class Reference	25
5.6.1 Detailed Description	26
5.6.2 Constructor & Destructor Documentation	26
5.6.2.1 BasicSolenoidSet()	26
5.6.3 Member Function Documentation	26
5.6.3.1 run()	26
5.7 BasicSpinCommand Class Reference	27
5.7.1 Detailed Description	28
5.7.2 Constructor & Destructor Documentation	28
5.7.2.1 BasicSpinCommand()	28
5.7.3 Member Function Documentation	29
5.7.3.1 run()	29
5.8 BasicStopCommand Class Reference	29
5.8.1 Detailed Description	30
5.8.2 Constructor & Destructor Documentation	31
5.8.2.1 BasicStopCommand()	31
5.8.3 Member Function Documentation	31
5.8.3.1 run()	31
5.9 Branch Class Reference	32
5.9.1 Detailed Description	33
5.9.2 Member Function Documentation	33
5.9.2.1 on_timeout()	33
5.9.2.2 run()	33
5.10 screen::ButtonConfig Struct Reference	34
5.11 screen::ButtonWidget Class Reference	34
5.11.1 Detailed Description	34
5.11.2 Constructor & Destructor Documentation	34
5.11.2.1 ButtonWidget() [1/2]	34
5.11.2.2 ButtonWidget() [2/2]	35
5.11.3 Member Function Documentation	35
5.11.3.1 update()	35
5.12 screen::CheckboxConfig Struct Reference	35
5.13 CommandController Class Reference	36
5.13.1 Detailed Description	36
5.13.2 Constructor & Destructor Documentation	36
5.13.2.1 CommandController()	36
5.13.3 Member Function Documentation	36
5.13.3.1 add() [1/3]	36
5.13.3.2 add() [2/3]	37
5.13.3.3 add() [3/3]	37
5.13.3.4 add_cancel_func()	38

---

5.13.3.5 add_delay()	38
5.13.3.6 last_command_timed_out()	38
5.13.3.7 run()	38
5.14 Condition Class Reference	39
5.14.1 Detailed Description	39
5.15 CustomEncoder Class Reference	39
5.15.1 Detailed Description	40
5.15.2 Constructor & Destructor Documentation	40
5.15.2.1 CustomEncoder()	40
5.15.3 Member Function Documentation	41
5.15.3.1 position()	41
5.15.3.2 rotation()	41
5.15.3.3 setPosition()	41
5.15.3.4 setRotation()	42
5.15.3.5 velocity()	42
5.16 DelayCommand Class Reference	42
5.16.1 Detailed Description	43
5.16.2 Constructor & Destructor Documentation	44
5.16.2.1 DelayCommand()	44
5.16.3 Member Function Documentation	45
5.16.3.1 run()	45
5.17 DriveForwardCommand Class Reference	45
5.17.1 Detailed Description	46
5.17.2 Constructor & Destructor Documentation	47
5.17.2.1 DriveForwardCommand()	47
5.17.3 Member Function Documentation	47
5.17.3.1 on_timeout()	47
5.17.3.2 run()	48
5.18 DriveStopCommand Class Reference	48
5.18.1 Detailed Description	49
5.18.2 Constructor & Destructor Documentation	49
5.18.2.1 DriveStopCommand()	49
5.18.3 Member Function Documentation	49
5.18.3.1 on_timeout()	49
5.18.3.2 run()	50
5.19 DriveToPointCommand Class Reference	50
5.19.1 Detailed Description	51
5.19.2 Constructor & Destructor Documentation	52
5.19.2.1 DriveToPointCommand() [1/2]	52
5.19.2.2 DriveToPointCommand() [2/2]	52
5.19.3 Member Function Documentation	52
5.19.3.1 run()	52

5.20 AutoChooser::entry_t Struct Reference . . . . .	53
5.20.1 Detailed Description . . . . .	53
5.20.2 Member Data Documentation . . . . .	54
5.20.2.1 name . . . . .	54
5.21 ExponentialMovingAverage Class Reference . . . . .	54
5.21.1 Detailed Description . . . . .	55
5.21.2 Constructor & Destructor Documentation . . . . .	55
5.21.2.1 ExponentialMovingAverage() [1/2] . . . . .	55
5.21.2.2 ExponentialMovingAverage() [2/2] . . . . .	55
5.21.3 Member Function Documentation . . . . .	56
5.21.3.1 add_entry() . . . . .	56
5.21.3.2 get_size() . . . . .	56
5.21.3.3 get_value() . . . . .	56
5.22 Feedback Class Reference . . . . .	57
5.22.1 Detailed Description . . . . .	57
5.22.2 Member Function Documentation . . . . .	57
5.22.2.1 get() . . . . .	57
5.22.2.2 init() . . . . .	57
5.22.2.3 is_on_target() . . . . .	58
5.22.2.4 set_limits() . . . . .	58
5.22.2.5 update() . . . . .	58
5.23 FeedForward Class Reference . . . . .	59
5.23.1 Detailed Description . . . . .	59
5.23.2 Constructor & Destructor Documentation . . . . .	59
5.23.2.1 FeedForward() . . . . .	59
5.23.3 Member Function Documentation . . . . .	60
5.23.3.1 calculate() . . . . .	60
5.24 FeedForward::ff_config_t Struct Reference . . . . .	60
5.24.1 Detailed Description . . . . .	60
5.24.2 Member Data Documentation . . . . .	61
5.24.2.1 kA . . . . .	61
5.24.2.2 kG . . . . .	61
5.24.2.3 kS . . . . .	61
5.24.2.4 kV . . . . .	61
5.25 Filter Class Reference . . . . .	61
5.25.1 Detailed Description . . . . .	62
5.25.2 Member Function Documentation . . . . .	62
5.25.2.1 add_entry() . . . . .	62
5.25.2.2 get_value() . . . . .	62
5.26 Flywheel Class Reference . . . . .	62
5.26.1 Detailed Description . . . . .	63
5.26.2 Constructor & Destructor Documentation . . . . .	63

---

5.26.2.1 Flywheel()	63
5.26.3 Member Function Documentation	63
5.26.3.1 get_motors()	63
5.26.3.2 get_target()	64
5.26.3.3 getRPM()	64
5.26.3.4 is_on_target()	64
5.26.3.5 Page()	64
5.26.3.6 spin_manual()	64
5.26.3.7 spin_rpm()	65
5.26.3.8 SpinRpmCmd()	65
5.26.3.9 stop()	66
5.26.3.10 WaitUntilUpToSpeedCmd()	66
5.26.4 Friends And Related Symbol Documentation	66
5.26.4.1 spinRPMTask	66
5.27 FlywheelPage Class Reference	66
5.27.1 Member Function Documentation	67
5.27.1.1 draw()	67
5.27.1.2 update()	67
5.28 FlywheelStopCommand Class Reference	68
5.28.1 Detailed Description	69
5.28.2 Constructor & Destructor Documentation	69
5.28.2.1 FlywheelStopCommand()	69
5.28.3 Member Function Documentation	69
5.28.3.1 run()	69
5.29 FlywheelStopMotorsCommand Class Reference	70
5.29.1 Detailed Description	71
5.29.2 Constructor & Destructor Documentation	71
5.29.2.1 FlywheelStopMotorsCommand()	71
5.29.3 Member Function Documentation	71
5.29.3.1 run()	71
5.30 FlywheelStopNonTasksCommand Class Reference	72
5.30.1 Detailed Description	73
5.31 FunctionCommand Class Reference	73
5.31.1 Detailed Description	74
5.31.2 Member Function Documentation	74
5.31.2.1 run()	74
5.32 FunctionCondition Class Reference	75
5.32.1 Detailed Description	75
5.32.2 Member Function Documentation	76
5.32.2.1 test()	76
5.33 screen::FunctionPage Class Reference	76
5.33.1 Detailed Description	77

5.33.2 Constructor & Destructor Documentation . . . . .	77
5.33.2.1 FunctionPage() . . . . .	77
5.33.3 Member Function Documentation . . . . .	77
5.33.3.1 draw() . . . . .	77
5.33.3.2 update() . . . . .	78
5.34 GenericAuto Class Reference . . . . .	78
5.34.1 Detailed Description . . . . .	78
5.34.2 Member Function Documentation . . . . .	78
5.34.2.1 add() . . . . .	78
5.34.2.2 add_async() . . . . .	79
5.34.2.3 add_delay() . . . . .	79
5.34.2.4 run() . . . . .	79
5.35 GraphDrawer Class Reference . . . . .	80
5.35.1 Constructor & Destructor Documentation . . . . .	80
5.35.1.1 GraphDrawer() . . . . .	80
5.35.2 Member Function Documentation . . . . .	80
5.35.2.1 add_samples() [1/2] . . . . .	80
5.35.2.2 add_samples() [2/2] . . . . .	81
5.35.2.3 draw() . . . . .	81
5.36 PurePursuit::hermite_point Struct Reference . . . . .	81
5.36.1 Detailed Description . . . . .	82
5.37 IfTimePassed Class Reference . . . . .	82
5.37.1 Detailed Description . . . . .	83
5.37.2 Member Function Documentation . . . . .	83
5.37.2.1 test() . . . . .	83
5.38 InOrder Class Reference . . . . .	83
5.38.1 Detailed Description . . . . .	84
5.38.2 Member Function Documentation . . . . .	85
5.38.2.1 on_timeout() . . . . .	85
5.38.2.2 run() . . . . .	85
5.39 screen::LabelConfig Struct Reference . . . . .	85
5.40 Lift< T > Class Template Reference . . . . .	85
5.40.1 Detailed Description . . . . .	86
5.40.2 Constructor & Destructor Documentation . . . . .	86
5.40.2.1 Lift() . . . . .	86
5.40.3 Member Function Documentation . . . . .	87
5.40.3.1 control_continuous() . . . . .	87
5.40.3.2 control_manual() . . . . .	87
5.40.3.3 control_setpoints() . . . . .	87
5.40.3.4 get_async() . . . . .	88
5.40.3.5 get_setpoint() . . . . .	88
5.40.3.6 hold() . . . . .	88

---

5.40.3.7 <code>home()</code>	88
5.40.3.8 <code>set_async()</code>	88
5.40.3.9 <code>set_position()</code>	89
5.40.3.10 <code>set_sensor_function()</code>	89
5.40.3.11 <code>set_sensor_reset()</code>	89
5.40.3.12 <code>set_setpoint()</code>	89
5.41 <code>Lift&lt; T &gt;::lift_cfg_t</code> Struct Reference	90
5.41.1 Detailed Description	90
5.42 Logger Class Reference	91
5.42.1 Detailed Description	91
5.42.2 Constructor & Destructor Documentation	91
5.42.2.1 <code>Logger()</code>	91
5.42.3 Member Function Documentation	92
5.42.3.1 <code>Log()</code> [1/2]	92
5.42.3.2 <code>Log()</code> [2/2]	92
5.42.3.3 <code>Logf()</code> [1/2]	92
5.42.3.4 <code>Logf()</code> [2/2]	93
5.42.3.5 <code>Logln()</code> [1/2]	93
5.42.3.6 <code>Logln()</code> [2/2]	93
5.43 <code>MotionController::m_profile_cfg_t</code> Struct Reference	93
5.43.1 Detailed Description	94
5.44 <code>Mat2</code> Struct Reference	94
5.45 MecanumDrive Class Reference	95
5.45.1 Detailed Description	95
5.45.2 Constructor & Destructor Documentation	95
5.45.2.1 <code>MecanumDrive()</code>	95
5.45.3 Member Function Documentation	96
5.45.3.1 <code>auto_drive()</code>	96
5.45.3.2 <code>auto_turn()</code>	96
5.45.3.3 <code>drive()</code>	97
5.45.3.4 <code>drive_raw()</code>	98
5.46 <code>MecanumDrive::mecanumdrive_config_t</code> Struct Reference	98
5.46.1 Detailed Description	99
5.47 <code>motion_t</code> Struct Reference	99
5.47.1 Detailed Description	99
5.48 MotionController Class Reference	99
5.48.1 Detailed Description	101
5.48.2 Constructor & Destructor Documentation	101
5.48.2.1 <code>MotionController()</code>	101
5.48.3 Member Function Documentation	101
5.48.3.1 <code>get()</code>	101
5.48.3.2 <code>get_motion()</code>	102

5.48.3.3 init()	102
5.48.3.4 is_on_target()	102
5.48.3.5 set_limits()	102
5.48.3.6 tune_feedforward()	103
5.48.3.7 update()	103
5.49 MotionControllerPage Class Reference	104
5.49.1 Member Function Documentation	105
5.49.1.1 draw()	105
5.49.1.2 update()	105
5.50 MovingAverage Class Reference	106
5.50.1 Detailed Description	106
5.50.2 Constructor & Destructor Documentation	107
5.50.2.1 MovingAverage() [1/2]	107
5.50.2.2 MovingAverage() [2/2]	107
5.50.3 Member Function Documentation	107
5.50.3.1 add_entry()	107
5.50.3.2 get_size()	107
5.50.3.3 get_value()	108
5.51 Odometry3Wheel Class Reference	108
5.51.1 Detailed Description	110
5.51.2 Constructor & Destructor Documentation	110
5.51.2.1 Odometry3Wheel()	110
5.51.3 Member Function Documentation	111
5.51.3.1 tune()	111
5.51.3.2 update()	111
5.52 Odometry3Wheel::odometry3wheel_cfg_t Struct Reference	112
5.52.1 Detailed Description	112
5.52.2 Member Data Documentation	112
5.52.2.1 off_axis_center_dist	112
5.52.2.2 wheel_diam	112
5.52.2.3 wheelbase_dist	112
5.53 OdometryBase Class Reference	113
5.53.1 Detailed Description	114
5.53.2 Constructor & Destructor Documentation	114
5.53.2.1 OdometryBase()	114
5.53.3 Member Function Documentation	115
5.53.3.1 background_task()	115
5.53.3.2 end_async()	115
5.53.3.3 get_accel()	115
5.53.3.4 get_angular_accel_deg()	115
5.53.3.5 get_angular_speed_deg()	116
5.53.3.6 get_position()	116

---

5.53.3.7 get_speed()	116
5.53.3.8 pos_diff()	116
5.53.3.9 rot_diff()	117
5.53.3.10 set_position()	117
5.53.3.11 smallest_angle()	117
5.53.3.12 update()	118
5.53.4 Member Data Documentation	118
5.53.4.1 accel	118
5.53.4.2 ang_accel_deg	118
5.53.4.3 ang_speed_deg	118
5.53.4.4 current_pos	119
5.53.4.5 handle	119
5.53.4.6 mut	119
5.53.4.7 speed	119
5.53.4.8 zero_pos	119
5.54 screen::OdometryPage Class Reference	120
5.54.1 Detailed Description	120
5.54.2 Constructor & Destructor Documentation	121
5.54.2.1 OdometryPage()	121
5.54.3 Member Function Documentation	121
5.54.3.1 draw()	121
5.54.3.2 update()	121
5.55 OdometryTank Class Reference	122
5.55.1 Detailed Description	123
5.55.2 Constructor & Destructor Documentation	123
5.55.2.1 OdometryTank() [1/3]	123
5.55.2.2 OdometryTank() [2/3]	124
5.55.2.3 OdometryTank() [3/3]	124
5.55.3 Member Function Documentation	125
5.55.3.1 set_position()	125
5.55.3.2 update()	125
5.56 OdomSetPosition Class Reference	125
5.56.1 Detailed Description	126
5.56.2 Constructor & Destructor Documentation	127
5.56.2.1 OdomSetPosition()	127
5.56.3 Member Function Documentation	128
5.56.3.1 run()	128
5.57 OrCondition Class Reference	128
5.57.1 Member Function Documentation	129
5.57.1.1 test()	129
5.58 screen::Page Class Reference	129
5.58.1 Detailed Description	130

---

5.58.2 Member Function Documentation . . . . .	130
5.58.2.1 draw() . . . . .	130
5.58.2.2 update() . . . . .	131
5.59 Parallel Class Reference . . . . .	131
5.59.1 Detailed Description . . . . .	132
5.59.2 Member Function Documentation . . . . .	133
5.59.2.1 on_timeout() . . . . .	133
5.59.2.2 run() . . . . .	133
5.60 parallel_runner_info Struct Reference . . . . .	133
5.61 PurePursuit::Path Class Reference . . . . .	134
5.61.1 Detailed Description . . . . .	134
5.61.2 Constructor & Destructor Documentation . . . . .	134
5.61.2.1 Path() . . . . .	134
5.61.3 Member Function Documentation . . . . .	134
5.61.3.1 get_points() . . . . .	134
5.61.3.2 get_radius() . . . . .	135
5.61.3.3 is_valid() . . . . .	135
5.62 PID Class Reference . . . . .	135
5.62.1 Detailed Description . . . . .	136
5.62.2 Member Enumeration Documentation . . . . .	136
5.62.2.1 ERROR_TYPE . . . . .	136
5.62.3 Constructor & Destructor Documentation . . . . .	136
5.62.3.1 PID() . . . . .	136
5.62.4 Member Function Documentation . . . . .	137
5.62.4.1 get() . . . . .	137
5.62.4.2 get_error() . . . . .	137
5.62.4.3 get_sensor_val() . . . . .	137
5.62.4.4 get_target() . . . . .	138
5.62.4.5 init() . . . . .	138
5.62.4.6 is_on_target() . . . . .	138
5.62.4.7 reset() . . . . .	138
5.62.4.8 set_limits() . . . . .	139
5.62.4.9 set_target() . . . . .	139
5.62.4.10 update() . . . . .	139
5.62.5 Member Data Documentation . . . . .	140
5.62.5.1 config . . . . .	140
5.63 PID::pid_config_t Struct Reference . . . . .	140
5.63.1 Detailed Description . . . . .	140
5.63.2 Member Data Documentation . . . . .	140
5.63.2.1 error_method . . . . .	140
5.63.2.2 on_target_time . . . . .	141
5.64 PIDFF Class Reference . . . . .	141

---

5.64.1 Member Function Documentation . . . . .	142
5.64.1.1 get() . . . . .	142
5.64.1.2 init() . . . . .	142
5.64.1.3 is_on_target() . . . . .	143
5.64.1.4 set_limits() . . . . .	143
5.64.1.5 set_target() . . . . .	143
5.64.1.6 update() [1/2] . . . . .	144
5.64.1.7 update() [2/2] . . . . .	144
5.65 screen::PIDPage Class Reference . . . . .	144
5.65.1 Detailed Description . . . . .	145
5.65.2 Constructor & Destructor Documentation . . . . .	145
5.65.2.1 PIDPage() . . . . .	145
5.65.3 Member Function Documentation . . . . .	146
5.65.3.1 draw() . . . . .	146
5.65.3.2 update() . . . . .	146
5.66 point_t Struct Reference . . . . .	146
5.66.1 Detailed Description . . . . .	147
5.66.2 Member Function Documentation . . . . .	147
5.66.2.1 dist() . . . . .	147
5.66.2.2 operator+() . . . . .	147
5.66.2.3 operator-() . . . . .	148
5.67 pose_t Struct Reference . . . . .	148
5.67.1 Detailed Description . . . . .	149
5.68 PurePursuitCommand Class Reference . . . . .	149
5.68.1 Detailed Description . . . . .	150
5.68.2 Constructor & Destructor Documentation . . . . .	150
5.68.2.1 PurePursuitCommand() . . . . .	150
5.68.3 Member Function Documentation . . . . .	151
5.68.3.1 on_timeout() . . . . .	151
5.68.3.2 run() . . . . .	151
5.69 Rect Struct Reference . . . . .	151
5.70 RepeatUntil Class Reference . . . . .	152
5.70.1 Constructor & Destructor Documentation . . . . .	153
5.70.1.1 RepeatUntil() [1/2] . . . . .	153
5.70.1.2 RepeatUntil() [2/2] . . . . .	153
5.70.2 Member Function Documentation . . . . .	154
5.70.2.1 on_timeout() . . . . .	154
5.70.2.2 run() . . . . .	154
5.71 robot_specs_t Struct Reference . . . . .	154
5.71.1 Detailed Description . . . . .	155
5.72 screen::ScreenData Struct Reference . . . . .	155
5.72.1 Detailed Description . . . . .	156

5.73 screen::ScreenRect Struct Reference . . . . .	156
5.74 Serializer Class Reference . . . . .	156
5.74.1 Detailed Description . . . . .	157
5.74.2 Constructor & Destructor Documentation . . . . .	157
5.74.2.1 Serializer() . . . . .	157
5.74.3 Member Function Documentation . . . . .	157
5.74.3.1 bool_or() . . . . .	157
5.74.3.2 double_or() . . . . .	157
5.74.3.3 int_or() . . . . .	158
5.74.3.4 save_to_disk() . . . . .	158
5.74.3.5 set_bool() . . . . .	158
5.74.3.6 set_double() . . . . .	159
5.74.3.7 set_int() . . . . .	159
5.74.3.8 set_string() . . . . .	159
5.74.3.9 string_or() . . . . .	159
5.75 screen::SizedWidget Struct Reference . . . . .	160
5.76 SliderCfg Struct Reference . . . . .	161
5.77 screen::SliderConfig Struct Reference . . . . .	161
5.78 screen::SliderWidget Class Reference . . . . .	161
5.78.1 Detailed Description . . . . .	161
5.78.2 Constructor & Destructor Documentation . . . . .	162
5.78.2.1 SliderWidget() . . . . .	162
5.78.3 Member Function Documentation . . . . .	163
5.78.3.1 update() . . . . .	163
5.79 SpinRPMCommand Class Reference . . . . .	163
5.79.1 Detailed Description . . . . .	165
5.79.2 Constructor & Destructor Documentation . . . . .	165
5.79.2.1 SpinRPMCommand() . . . . .	165
5.79.3 Member Function Documentation . . . . .	165
5.79.3.1 run() . . . . .	165
5.80 PurePursuit::spline Struct Reference . . . . .	166
5.80.1 Detailed Description . . . . .	166
5.81 screen::StatsPage Class Reference . . . . .	166
5.81.1 Detailed Description . . . . .	167
5.81.2 Constructor & Destructor Documentation . . . . .	167
5.81.2.1 StatsPage() . . . . .	167
5.81.3 Member Function Documentation . . . . .	167
5.81.3.1 draw() . . . . .	167
5.81.3.2 update() . . . . .	168
5.82 TakeBackHalf Class Reference . . . . .	168
5.82.1 Detailed Description . . . . .	169
5.82.2 Member Function Documentation . . . . .	169

---

5.82.2.1 get()	169
5.82.2.2 init()	170
5.82.2.3 is_on_target()	170
5.82.2.4 set_limits()	170
5.82.2.5 update()	170
5.83 TankDrive Class Reference	172
5.83.1 Detailed Description	173
5.83.2 Member Enumeration Documentation	173
5.83.2.1 BrakeType	173
5.83.3 Constructor & Destructor Documentation	173
5.83.3.1 TankDrive()	173
5.83.4 Member Function Documentation	174
5.83.4.1 drive_arena()	174
5.83.4.2 drive_forward() [1/2]	174
5.83.4.3 drive_forward() [2/2]	175
5.83.4.4 drive_tank()	176
5.83.4.5 drive_tank_raw()	176
5.83.4.6 drive_to_point() [1/2]	176
5.83.4.7 drive_to_point() [2/2]	177
5.83.4.8 modify_inputs()	178
5.83.4.9 pure_pursuit() [1/2]	178
5.83.4.10 pure_pursuit() [2/2]	179
5.83.4.11 reset_auto()	180
5.83.4.12 stop()	180
5.83.4.13 turn_degrees() [1/2]	180
5.83.4.14 turn_degrees() [2/2]	181
5.83.4.15 turn_to_heading() [1/2]	182
5.83.4.16 turn_to_heading() [2/2]	182
5.84 screen::TextConfig Struct Reference	183
5.85 TimesTestedCondition Class Reference	183
5.85.1 Member Function Documentation	184
5.85.1.1 test()	184
5.86 trapezoid_profile_segment_t Struct Reference	184
5.86.1 Detailed Description	185
5.87 TrapezoidProfile Class Reference	185
5.87.1 Detailed Description	185
5.87.2 Constructor & Destructor Documentation	186
5.87.2.1 TrapezoidProfile()	186
5.87.3 Member Function Documentation	186
5.87.3.1 calculate()	186
5.87.3.2 calculate_time_based()	188
5.87.3.3 get_movement_time()	188

5.87.3.4 set_accel() . . . . .	188
5.87.3.5 set_endpts() . . . . .	189
5.87.3.6 set_max_v() . . . . .	189
5.87.3.7 set_vel_endpts() . . . . .	189
5.88 TurnDegreesCommand Class Reference . . . . .	189
5.88.1 Detailed Description . . . . .	191
5.88.2 Constructor & Destructor Documentation . . . . .	191
5.88.2.1 TurnDegreesCommand() . . . . .	191
5.88.3 Member Function Documentation . . . . .	191
5.88.3.1 on_timeout() . . . . .	191
5.88.3.2 run() . . . . .	192
5.89 TurnToHeadingCommand Class Reference . . . . .	192
5.89.1 Detailed Description . . . . .	193
5.89.2 Constructor & Destructor Documentation . . . . .	193
5.89.2.1 TurnToHeadingCommand() . . . . .	193
5.89.3 Member Function Documentation . . . . .	194
5.89.3.1 on_timeout() . . . . .	194
5.89.3.2 run() . . . . .	194
5.90 Vector2D Class Reference . . . . .	194
5.90.1 Detailed Description . . . . .	195
5.90.2 Constructor & Destructor Documentation . . . . .	195
5.90.2.1 Vector2D() [1/2] . . . . .	195
5.90.2.2 Vector2D() [2/2] . . . . .	196
5.90.3 Member Function Documentation . . . . .	196
5.90.3.1 get_dir() . . . . .	196
5.90.3.2 get_mag() . . . . .	196
5.90.3.3 get_x() . . . . .	197
5.90.3.4 get_y() . . . . .	197
5.90.3.5 normalize() . . . . .	197
5.90.3.6 operator*() . . . . .	197
5.90.3.7 operator+() . . . . .	198
5.90.3.8 operator-() . . . . .	198
5.90.3.9 point() . . . . .	198
5.91 WaitUntilCondition Class Reference . . . . .	199
5.91.1 Detailed Description . . . . .	200
5.91.2 Member Function Documentation . . . . .	200
5.91.2.1 run() . . . . .	200
5.92 WaitUntilUpToSpeedCommand Class Reference . . . . .	201
5.92.1 Detailed Description . . . . .	202
5.92.2 Constructor & Destructor Documentation . . . . .	202
5.92.2.1 WaitUntilUpToSpeedCommand() . . . . .	202
5.92.3 Member Function Documentation . . . . .	202

---

5.92.3.1 run() . . . . .	202
5.93 screen::WidgetConfig Struct Reference . . . . .	203
5.94 screen::WidgetPage Class Reference . . . . .	204
5.94.1 Member Function Documentation . . . . .	204
5.94.1.1 draw() . . . . .	204
5.94.1.2 update() . . . . .	205
<b>6 File Documentation</b> . . . . .	<b>207</b>
6.1 robot_specs.h . . . . .	207
6.2 custom_encoder.h . . . . .	207
6.3 flywheel.h . . . . .	208
6.4 layout.h . . . . .	208
6.5 lift.h . . . . .	209
6.6 mecanum_drive.h . . . . .	211
6.7 odometry_3wheel.h . . . . .	212
6.8 odometry_base.h . . . . .	213
6.9 odometry_tank.h . . . . .	213
6.10 screen.h . . . . .	214
6.11 tank_drive.h . . . . .	217
6.12 auto_chooser.h . . . . .	218
6.13 auto_command.h . . . . .	219
6.14 basic_command.h . . . . .	221
6.15 command_controller.h . . . . .	221
6.16 delay_command.h . . . . .	222
6.17 drive_commands.h . . . . .	222
6.18 flywheel_commands.h . . . . .	224
6.19 bang_bang.h . . . . .	225
6.20 feedback_base.h . . . . .	225
6.21 feedforward.h . . . . .	226
6.22 motion_controller.h . . . . .	226
6.23 pid.h . . . . .	227
6.24 pidff.h . . . . .	228
6.25 take_back_half.h . . . . .	228
6.26 trapezoid_profile.h . . . . .	229
6.27 generic_auto.h . . . . .	230
6.28 geometry.h . . . . .	230
6.29 graph_drawer.h . . . . .	231
6.30 logger.h . . . . .	232
6.31 math_util.h . . . . .	232
6.32 moving_average.h . . . . .	233
6.33 pure_pursuit.h . . . . .	234
6.34 serializer.h . . . . .	235

6.35 vector2d.h . . . . .	236
<b>Index</b>	<b>237</b>

# 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 . . . . .	20
Async . . . . .	14
BasicSolenoidSet . . . . .	25
BasicSpinCommand . . . . .	27
BasicStopCommand . . . . .	29
Branch . . . . .	32
DelayCommand . . . . .	42
DriveForwardCommand . . . . .	45
DriveStopCommand . . . . .	48
DriveToPointCommand . . . . .	50
FlywheelStopCommand . . . . .	68
FlywheelStopMotorsCommand . . . . .	70
FlywheelStopNonTasksCommand . . . . .	72
FunctionCommand . . . . .	73
InOrder . . . . .	83
OdomSetPosition . . . . .	125
Parallel . . . . .	131
PurePursuitCommand . . . . .	149
RepeatUntil . . . . .	152
SpinRPMCommand . . . . .	163
TurnDegreesCommand . . . . .	189
TurnToHeadingCommand . . . . .	192
WaitUntilCondition . . . . .	199
WaitUntilUpToSpeedCommand . . . . .	201
screen::ButtonConfig . . . . .	34
screen::ButtonWidget . . . . .	34
screen::CheckboxConfig . . . . .	35
CommandController . . . . .	36
Condition . . . . .	39
AndCondition . . . . .	13
FunctionCondition . . . . .	75
IfTimePassed . . . . .	82
OrCondition . . . . .	128
TimesTestedCondition . . . . .	183
vex::encoder	

CustomEncoder . . . . .	39
AutoChooser::entry_t . . . . .	53
Feedback . . . . .	57
BangBang . . . . .	22
MotionController . . . . .	99
PID . . . . .	135
PIDFF . . . . .	141
TakeBackHalf . . . . .	168
FeedForward . . . . .	59
FeedForward::ff_config_t . . . . .	60
Filter . . . . .	61
ExponentialMovingAverage . . . . .	54
MovingAverage . . . . .	106
Flywheel . . . . .	62
GenericAuto . . . . .	78
GraphDrawer . . . . .	80
PurePursuit::hermite_point . . . . .	81
screen::LabelConfig . . . . .	85
Lift< T > . . . . .	85
Lift< T >::lift_cfg_t . . . . .	90
Logger . . . . .	91
MotionController::m_profile_cfg_t . . . . .	93
Mat2 . . . . .	94
MecanumDrive . . . . .	95
MecanumDrive::mecanumdrive_config_t . . . . .	98
motion_t . . . . .	99
Odometry3Wheel::odometry3wheel_cfg_t . . . . .	112
OdometryBase . . . . .	113
Odometry3Wheel . . . . .	108
OdometryTank . . . . .	122
screen::Page . . . . .	129
AutoChooser . . . . .	16
FlywheelPage . . . . .	66
MotionControllerPage . . . . .	104
screen::FunctionPage . . . . .	76
screen::OdometryPage . . . . .	120
screen::PIDPage . . . . .	144
screen::StatsPage . . . . .	166
screen::WidgetPage . . . . .	204
parallel_runner_info . . . . .	133
PurePursuit::Path . . . . .	134
PID::pid_config_t . . . . .	140
point_t . . . . .	146
pose_t . . . . .	148
Rect . . . . .	151
robot_specs_t . . . . .	154
screen::ScreenData . . . . .	155
screen::ScreenRect . . . . .	156
Serializer . . . . .	156
screen::SizedWidget . . . . .	160
SliderCfg . . . . .	161
screen::SliderConfig . . . . .	161
screen::SliderWidget . . . . .	161
PurePursuit::spline . . . . .	166
TankDrive . . . . .	172
screen::TextConfig . . . . .	183
trapezoid_profile_segment_t . . . . .	184

TrapezoidProfile . . . . .	185
Vector2D . . . . .	194
screen::WidgetConfig . . . . .	203



# Chapter 3

## Class Index

### 3.1 Class List

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

AndCondition . . . . .	13
Async	
<b>Async</b> 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 . . . . .	16
AutoCommand . . . . .	20
BangBang . . . . .	22
BasicSolenoidSet . . . . .	25
BasicSpinCommand . . . . .	27
BasicStopCommand . . . . .	29
Branch	
<b>Branch</b> 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 . . . . .	32
screen::ButtonConfig . . . . .	34
screen::ButtonWidget	
Widget that does something when you tap it. The function is only called once when you first tap it	34
screen::CheckboxConfig . . . . .	35
CommandController . . . . .	36
Condition . . . . .	39
CustomEncoder . . . . .	39
DelayCommand . . . . .	42
DriveForwardCommand . . . . .	45
DriveStopCommand . . . . .	48
DriveToPointCommand . . . . .	50
AutoChooser::entry_t . . . . .	53
ExponentialMovingAverage . . . . .	54
Feedback . . . . .	57
FeedForward . . . . .	59
FeedForward::ff_config_t . . . . .	60
Filter . . . . .	61
Flywheel . . . . .	62
FlywheelPage . . . . .	66
FlywheelStopCommand . . . . .	68
FlywheelStopMotorsCommand . . . . .	70

FlywheelStopNonTasksCommand	72
FunctionCommand	73
FunctionCondition	
FunctionCondition is a quick and dirty <a href="#">Condition</a> to wrap some expression that should be evaluated at runtime	75
screen::FunctionPage	
Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything	76
GenericAuto	78
GraphDrawer	80
PurePursuit::hermite_point	81
IfTimePassed	
IfTimePassed tests based on time since the command controller was constructed. Returns true if elapsed time > time_s	82
InOrder	
InOrder runs its commands sequentially then continues. How to handle timeout in this case.	
Automatically set it to sum of commands timeouts?	83
screen::LabelConfig	85
Lift< T >	85
Lift< T >::lift_cfg_t	90
Logger	
Class to simplify writing to files	91
MotionController::m_profile_cfg_t	93
Mat2	94
MecanumDrive	95
MecanumDrive::mecanumdrive_config_t	98
motion_t	99
MotionController	99
MotionControllerPage	104
MovingAverage	106
Odometry3Wheel	108
Odometry3Wheel::odometry3wheel_cfg_t	112
OdometryBase	113
screen::OdometryPage	
Page that shows odometry position and rotation and a map (if an sd card with the file is on)	120
OdometryTank	122
OdomSetPosition	125
OrCondition	128
screen::Page	
Page describes one part of the screen slideshow	129
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	131
parallel_runner_info	133
PurePursuit::Path	134
PID	135
PID::pid_config_t	140
PIDFF	141
screen::PIDPage	
PIDPage provides a way to tune a pid controller on the screen	144
point_t	146
pose_t	148
PurePursuitCommand	149
Rect	151
RepeatUntil	152
robot_specs_t	154
screen::ScreenData	
Holds the data that will be passed to the screen thread you probably shouldnt have to use it	155

screen::ScreenRect . . . . .	156
Serializer	
Serializes Arbitrary data to a file on the SD Card . . . . .	156
screen::SizedWidget . . . . .	160
SliderCfg . . . . .	161
screen::SliderConfig . . . . .	161
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 . . . . .	161
SpinRPMCommand . . . . .	163
PurePursuit::spline . . . . .	166
screen::StatsPage	
Draws motor stats and battery stats to the screen . . . . .	166
TakeBackHalf	
A velocity controller . . . . .	168
TankDrive . . . . .	172
screen::TextConfig . . . . .	183
TimesTestedCondition . . . . .	183
trapezoid_profile_segment_t . . . . .	184
TrapezoidProfile . . . . .	185
TurnDegreesCommand . . . . .	189
TurnToHeadingCommand . . . . .	192
Vector2D . . . . .	194
WaitUntilCondition	
Waits until the condition is true . . . . .	199
WaitUntilUpToSpeedCommand . . . . .	201
screen::WidgetConfig . . . . .	203
screen::WidgetPage . . . . .	204



# Chapter 4

## File Index

### 4.1 File List

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

include/robot_specs.h . . . . .	207
include/subsystems/custom_encoder.h . . . . .	207
include/subsystems/flywheel.h . . . . .	208
include/subsystems/layout.h . . . . .	208
include/subsystems/lift.h . . . . .	209
include/subsystems/mecanum_drive.h . . . . .	211
include/subsystems/screen.h . . . . .	214
include/subsystems/tank_drive.h . . . . .	217
include/subsystems/odometry/odometry_3wheel.h . . . . .	212
include/subsystems/odometry/odometry_base.h . . . . .	213
include/subsystems/odometry/odometry_tank.h . . . . .	213
include/utils/auto_chooser.h . . . . .	218
include/utils/generic_auto.h . . . . .	230
include/utils/geometry.h . . . . .	230
include/utils/graph_drawer.h . . . . .	231
include/utils/logger.h . . . . .	232
include/utils/math_util.h . . . . .	232
include/utils/moving_average.h . . . . .	233
include/utils/pure_pursuit.h . . . . .	234
include/utils/serializer.h . . . . .	235
include/utils/vector2d.h . . . . .	236
include/utils/command_structure/auto_command.h . . . . .	219
include/utils/command_structure/basic_command.h . . . . .	221
include/utils/command_structure/command_controller.h . . . . .	221
include/utils/command_structure/delay_command.h . . . . .	222
include/utils/command_structure/drive_commands.h . . . . .	222
include/utils/command_structure/flywheel_commands.h . . . . .	224
include/utils/controls/bang_bang.h . . . . .	225
include/utils/controls/feedback_base.h . . . . .	225
include/utils/controls/feedforward.h . . . . .	226
include/utils/controls/motion_controller.h . . . . .	226
include/utils/controls/pid.h . . . . .	227
include/utils/controls/pidff.h . . . . .	228
include/utils/controls/take_back_half.h . . . . .	228
include/utils/controls/trapezoid_profile.h . . . . .	229

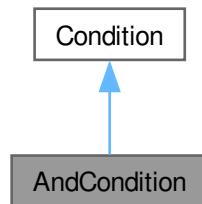


# Chapter 5

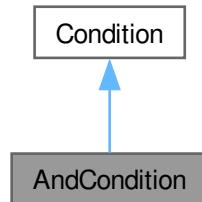
## Class Documentation

### 5.1 AndCondition Class Reference

Inheritance diagram for AndCondition:



Collaboration 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:

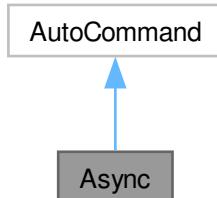
- [src/utils/command\\_structure/auto\\_command.cpp](#)

## 5.2 Async Class Reference

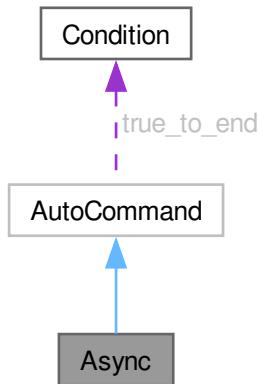
[Async](#) runs a command asynchronously will simply let it go and never look back THIS HAS A VERY NICHE USE CASE. THINK ABOUT IF YOU REALLY NEED IT.

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

Inheritance diagram for Async:



Collaboration 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](#).

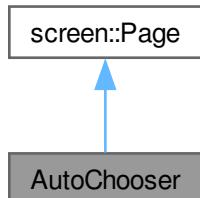
The documentation for this class was generated from the following files:

- include/utils/command\_structure/auto\_command.h
- src/utils/command\_structure/auto\_command.cpp

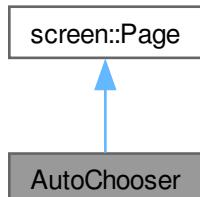
## 5.3 AutoChooser Class Reference

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

Inheritance diagram for AutoChooser:



Collaboration 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

<code>brain</code>	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]
```

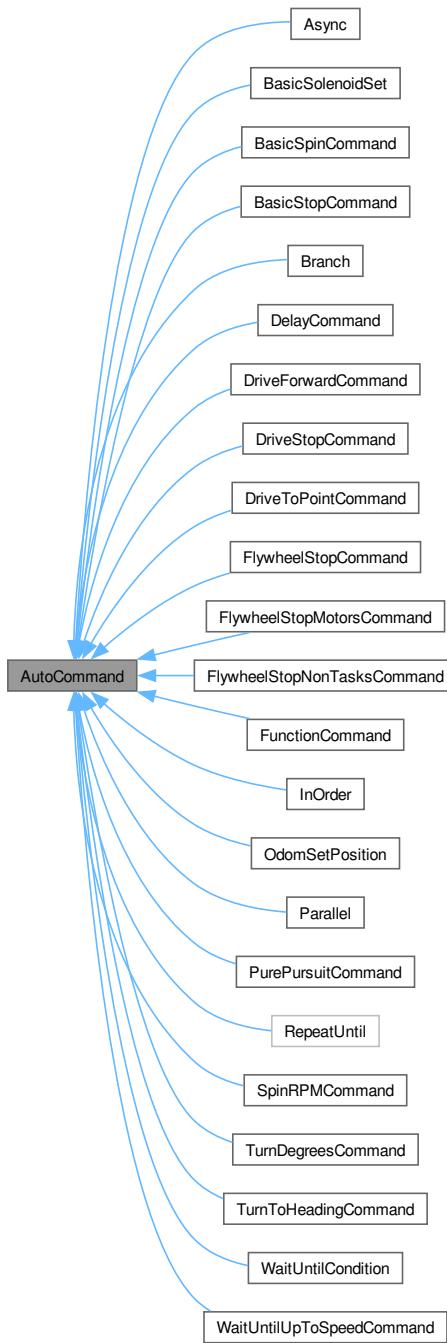
< a list of all possible auto choices

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

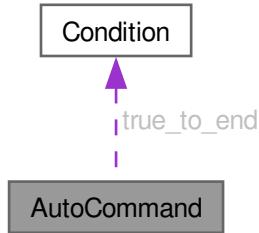
- include/utils/auto\_chooser.h
- src/utils/auto\_chooser.cpp

## 5.4 AutoCommand Class Reference

Inheritance diagram for AutoCommand:



Collaboration 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

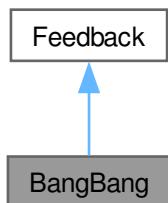
- a drive command hitting a wall and not being able to reach its target
- a command that waits until something is up to speed that never gets up to speed because of battery voltage
- something else...

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

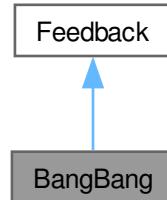
- [include/utils/command\\_structure/auto\\_command.h](#)

## 5.5 BangBang Class Reference

Inheritance diagram for BangBang:



Collaboration 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> Generated by Doxygen	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](#).

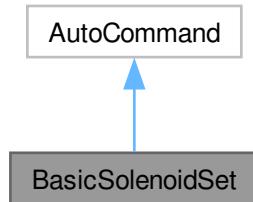
The documentation for this class was generated from the following files:

- include/utils/controls/bang\_bang.h
- src/utils/controls/bang\_bang.cpp

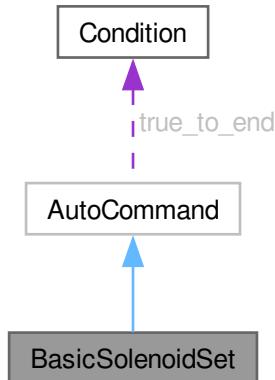
## 5.6 BasicSolenoidSet Class Reference

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

Inheritance diagram for BasicSolenoidSet:



Collaboration 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

<code>solenoid</code>	Solenoid being set
<code>setting</code>	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](#).

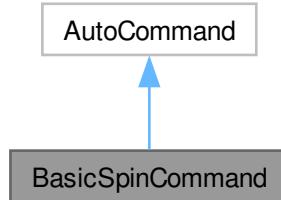
The documentation for this class was generated from the following files:

- include/utils/command\_structure/basic\_command.h
- src/utils/command\_structure/basic\_command.cpp

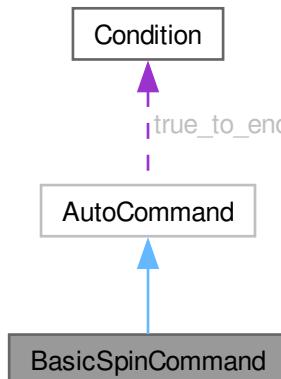
## 5.7 BasicSpinCommand Class Reference

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

Inheritance diagram for BasicSpinCommand:



Collaboration diagram for BasicSpinCommand:



### Public Types

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

### 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](#).

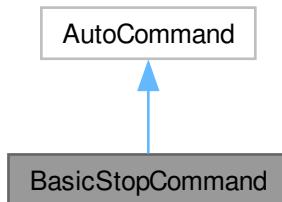
The documentation for this class was generated from the following files:

- include/utils/command\_structure/basic\_command.h
- src/utils/command\_structure/basic\_command.cpp

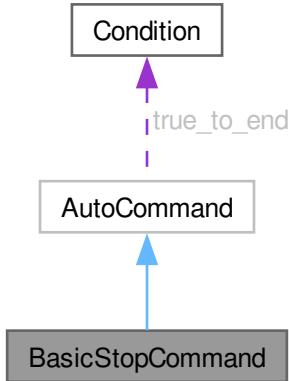
## 5.8 BasicStopCommand Class Reference

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

Inheritance diagram for BasicStopCommand:



Collaboration 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

<i>motor</i>	The motor to stop
<i>setting</i>	The brake setting for the motor
<i>motor</i>	Motor to stop
<i>setting</i>	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 Overrides run command from [AutoCommand](#).

#### Returns

True Command runs once

Reimplemented from [AutoCommand](#).

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

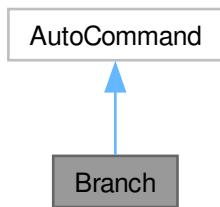
- include/utils/command\_structure/basic\_command.h
- src/utils/command\_structure/basic\_command.cpp

## 5.9 Branch Class Reference

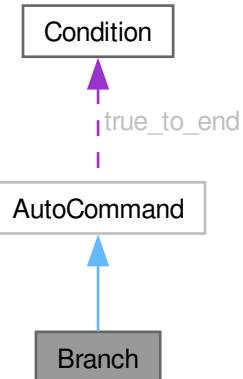
[Branch](#) chooses from multiple options at runtime. the function decider returns an index into the choices vector If you wish to make no choice and skip this section, return NO\_CHOICE; any choice that is out of bounds set to NO\_CHOICE.

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

Inheritance diagram for Branch:



Collaboration 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

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

<i>onpress</i>	the function to be called when the button is tapped
<i>rect</i>	the area the button should take up on the screen
<i>name</i>	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>	<input type="button" value=""/>
------------	---------------------------------

### 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 <a href="#">AutoCommand</a> 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

### 5.13.3.2 add() [2/3]

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

Adds a command to the queue

#### Parameters

<i>cmd</i>	the <a href="#">AutoCommand</a> 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>cmdss</i>	the AutoCommands we want to add to our list
--------------	---

### 5.13.3.3 add() [3/3]

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

Add multiple commands to the queue. No timeout here.

#### Parameters

<i>cmdss</i>	the AutoCommands we want to add to our list Add multiple commands to the queue. No timeout here.
<i>cmdss</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>cmdss</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

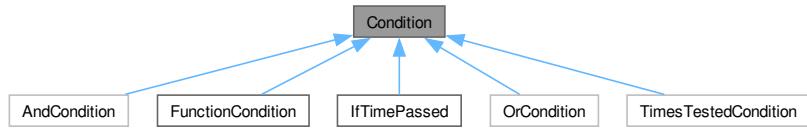
The documentation for this class was generated from the following files:

- 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

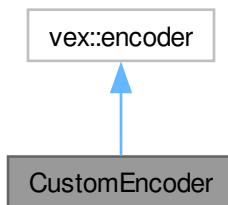
The documentation for this class was generated from the following files:

- include/utils/command\_structure/auto\_command.h
- src/utils/command\_structure/auto\_command.cpp

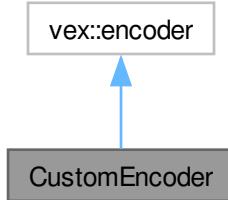
## 5.15 CustomEncoder Class Reference

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

Inheritance diagram for CustomEncoder:



Collaboration 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

<code>port</code>	the tripot port on the brain the encoder is plugged into
<code>ticks_per_rev</code>	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 <i>val</i>

#### 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

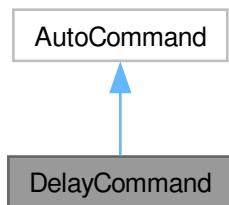
The documentation for this class was generated from the following files:

- include/subsystems/custom\_encoder.h
- src/subsystems/custom\_encoder.cpp

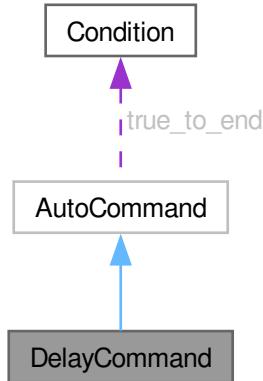
## 5.16 DelayCommand Class Reference

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

Inheritance diagram for DelayCommand:



Collaboration 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](#).

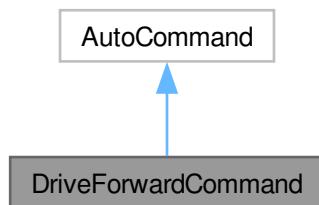
The documentation for this class was generated from the following file:

- include/utils/command\_structure/delay\_command.h

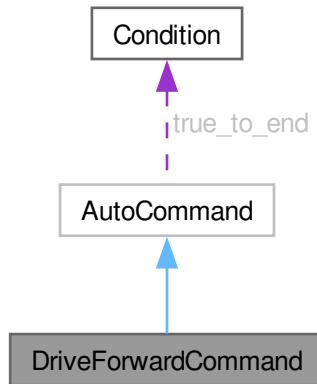
## 5.17 DriveForwardCommand Class Reference

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

Inheritance diagram for DriveForwardCommand:



Collaboration 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

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the drive
<i>inches</i>	how far forward to drive
<i>dir</i>	the direction to drive
<i>max_speed</i>	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](#).

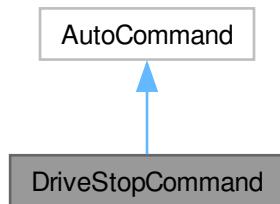
The documentation for this class was generated from the following files:

- include/utils/command\_structure/drive\_commands.h
- src/utils/command\_structure/drive\_commands.cpp

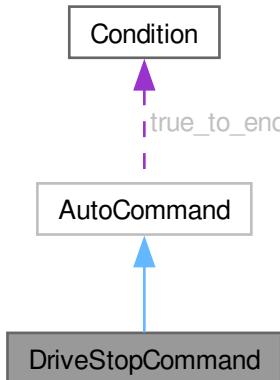
## 5.18 DriveStopCommand Class Reference

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

Inheritance diagram for DriveStopCommand:



Collaboration 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](#).

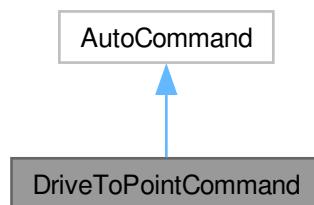
The documentation for this class was generated from the following files:

- include/utils/command\_structure/drive\_commands.h
- src/utils/command\_structure/drive\_commands.cpp

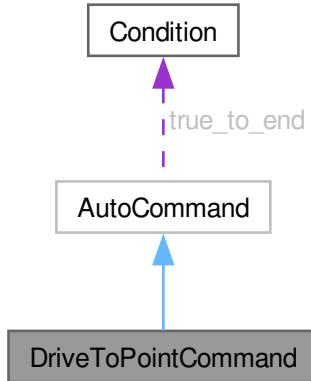
## 5.19 DriveToPointCommand Class Reference

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

Inheritance diagram for DriveToPointCommand:



Collaboration 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](#).

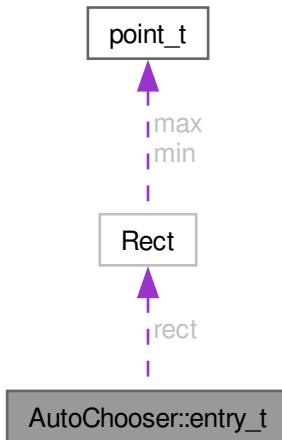
The documentation for this class was generated from the following files:

- 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>
```

Collaboration diagram for AutoChooser::entry\_t:



### 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 represented by the block

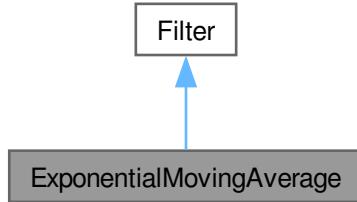
The documentation for this struct was generated from the following file:

- include/utils/auto\_chooser.h

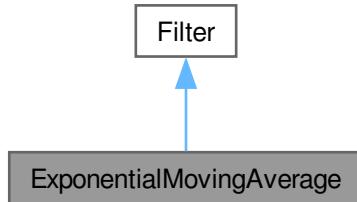
## 5.21 ExponentialMovingAverage Class Reference

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

Inheritance diagram for ExponentialMovingAverage:



Collaboration 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. sum(samples)/numsamples

How many samples the average is made from

##### Returns

the number of samples used to calculate this average

Implements [Filter](#).

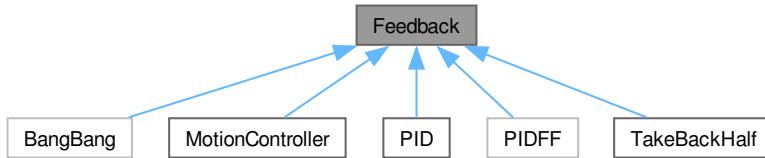
The documentation for this class was generated from the following files:

- include/utils/moving\_average.h
- src/utils/moving\_average.cpp

## 5.22 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**

<i>cfg</i>	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:  $F = kG + kS \operatorname{sgn}(v) + kV*v + kA*a$

**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.

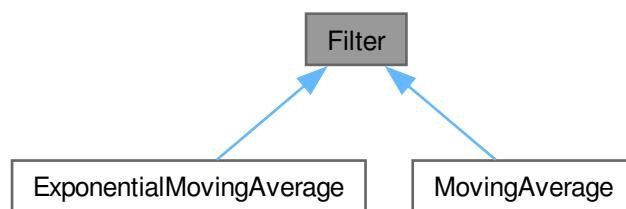
The documentation for this struct was generated from the following file:

- 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]
```

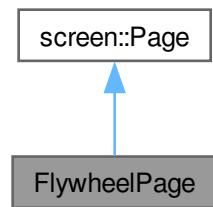
Runs a thread that keeps track of updating flywheel RPM and controlling it accordingly

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

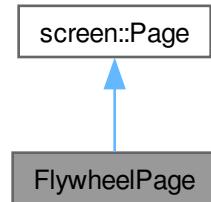
- include/subsystems/flywheel.h
- src/subsystems/flywheel.cpp

## 5.27 FlywheelPage Class Reference

Inheritance diagram for FlywheelPage:



Collaboration 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](#).

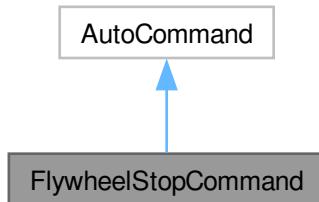
The documentation for this class was generated from the following file:

- src/subsystems/flywheel.cpp

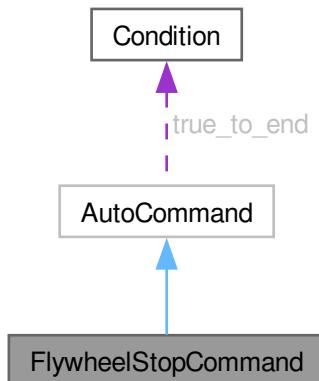
## 5.28 FlywheelStopCommand Class Reference

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

Inheritance diagram for FlywheelStopCommand:



Collaboration 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

<code>flywheel</code>	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](#).

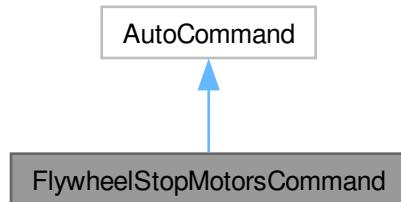
The documentation for this class was generated from the following files:

- include/utils/command\_structure/flywheel\_commands.h
- src/utils/command\_structure/flywheel\_commands.cpp

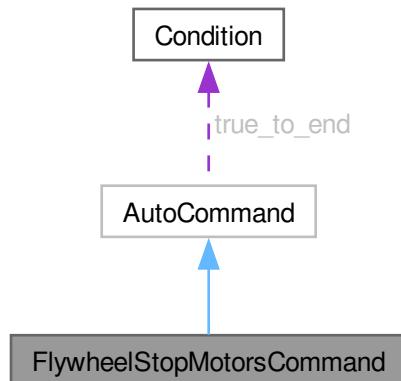
## 5.29 FlywheelStopMotorsCommand Class Reference

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

Inheritance diagram for FlywheelStopMotorsCommand:



Collaboration 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](#).

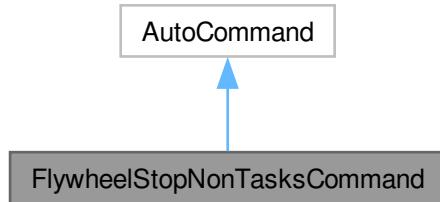
The documentation for this class was generated from the following files:

- include/utils/command\_structure/flywheel\_commands.h
- src/utils/command\_structure/flywheel\_commands.cpp

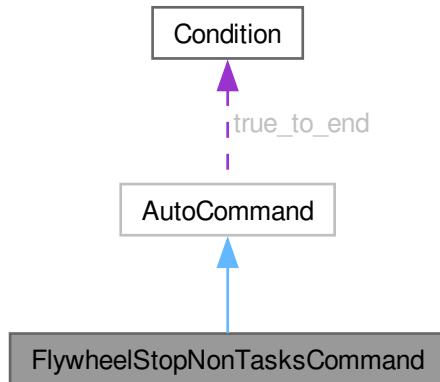
## 5.30 FlywheelStopNonTasksCommand Class Reference

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

Inheritance diagram for FlywheelStopNonTasksCommand:



Collaboration 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

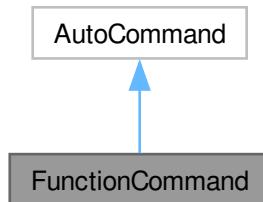
The documentation for this class was generated from the following files:

- include/utils/command\_structure/flywheel\_commands.h
- src/utils/command\_structure/flywheel\_commands.cpp

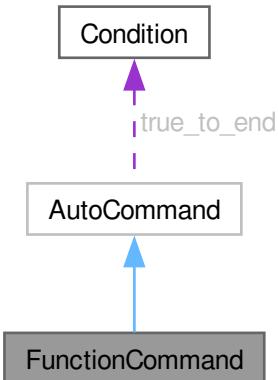
## 5.31 FunctionCommand Class Reference

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

Inheritance diagram for FunctionCommand:



Collaboration 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:

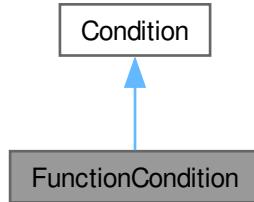
- include/utils/command\_structure/auto\_command.h

## 5.32 FunctionCondition Class Reference

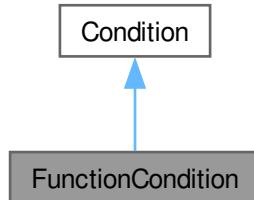
[FunctionCondition](#) is a quick and dirty [Condition](#) to wrap some expression that should be evaluated at runtime.

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

Inheritance diagram for FunctionCondition:



Collaboration 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:

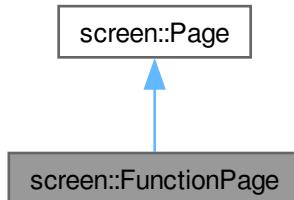
- include/utils/command\_structure/auto\_command.h
- src/utils/command\_structure/auto\_command.cpp

## 5.33 screen::FunctionPage Class Reference

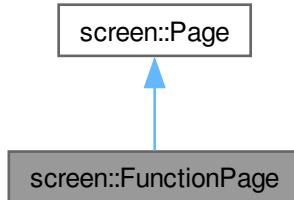
Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything.

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

Inheritance diagram for screen::FunctionPage:



Collaboration 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**

<i>new_state</i>	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**

<i>async_state</i>	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**

<i>ms</i>	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

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

<code>sample</code>	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
<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:

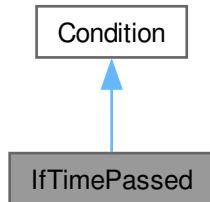
- include/utils/pure\_pursuit.h

## 5.37 IfTimePassed Class Reference

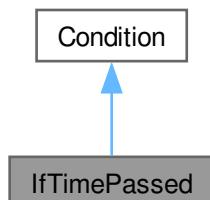
IfTimePassed tests based on time since the command controller was constructed. Returns true if elapsed time > time\_s.

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

Inheritance diagram for IfTimePassed:



Collaboration 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:

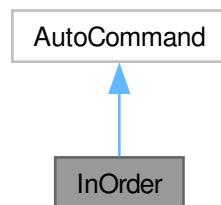
- `include/utils/command_structure/auto_command.h`
- `src/utils/command_structure/auto_command.cpp`

## 5.38 InOrder Class Reference

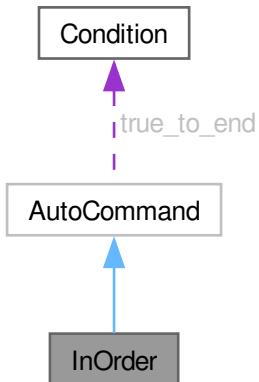
`InOrder` runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timeouts?

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

Inheritance diagram for InOrder:



Collaboration diagram for InOrder:



### Public Member Functions

- `InOrder (const InOrder &other)=default`
- `InOrder (std::queue< AutoCommand * > cmd)`
- `InOrder (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`

### 5.38.1 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; <code>PID</code> 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

<i>up_ctrl</i>	Button controlling the "UP" motion
<i>down_ctrl</i>	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

<code>val</code>	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>	<a href="#">Lift</a> setpoint, in motor revolutions or sensor units defined by <a href="#">get_sensor</a> . Cannot be outside the softstops.
------------	--

**Returns**

True if the pid has reached the setpoint

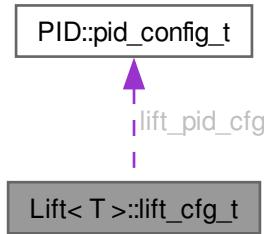
The documentation for this class was generated from the following file:

- include/subsystems/lift.h

## 5.41 Lift< T >::lift\_cfg\_t Struct Reference

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

Collaboration diagram for Lift< T >::lift\_cfg\_t:



### 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:

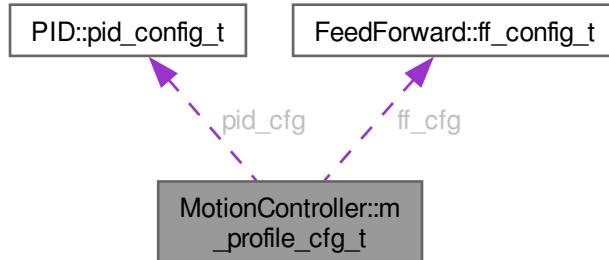
- include/utils/logger.h
- src/utils/logger.cpp

## 5.43 MotionController::m\_profile\_cfg\_t Struct Reference

---

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

Collaboration diagram for MotionController::m\_profile\_cfg\_t:



### 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 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)

### 5.45.1 Detailed Description

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

### 5.45.2 Constructor & Destructor Documentation

#### 5.45.2.1 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

### 5.45.3 Member Function Documentation

#### 5.45.3.1 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.

#### 5.45.3.2 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.

**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

**5.45.3.3 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)

#### 5.45.3.4 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

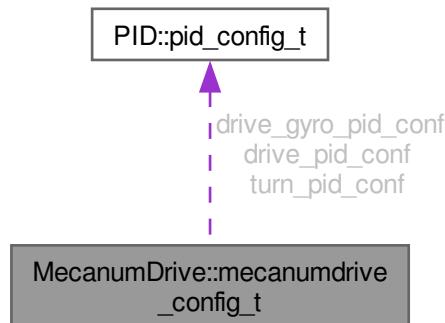
The documentation for this class was generated from the following files:

- include/subsystems/mecanum\_drive.h
- src/subsystems/mecanum\_drive.cpp

## 5.46 MecanumDrive::mecanumdrive\_config\_t Struct Reference

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

Collaboration diagram for MecanumDrive::mecanumdrive\_config\_t:



##### 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`

### 5.46.1 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

## 5.47 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*

### 5.47.1 Detailed Description

[motion\\_t](#) is a description of 1 dimensional motion at a point in time.

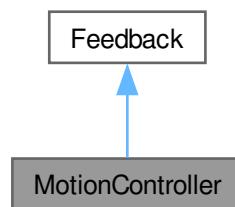
The documentation for this struct was generated from the following file:

- include/utils/controls/trapezoid\_profile.h

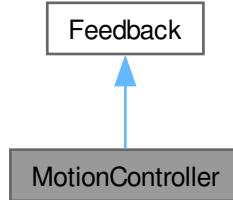
## 5.48 MotionController Class Reference

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

Inheritance diagram for MotionController:



Collaboration 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](#)

### 5.48.1 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

### 5.48.2 Constructor & Destructor Documentation

#### 5.48.2.1 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
---------------------	--

### 5.48.3 Member Function Documentation

#### 5.48.3.1 get()

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

Returns

the last saved result from the feedback controller

Implements [Feedback](#).

### 5.48.3.2 get\_motion()

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

#### Returns

The current position, velocity and acceleration setpoints

### 5.48.3.3 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](#).

### 5.48.3.4 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](#).

### 5.48.3.5 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](#).

**5.48.3.6 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\*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

**5.48.3.7 update()**

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

Update the motion profile with a new sensor value.

**Parameters**

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

**Returns**

the motor input generated from the motion profile

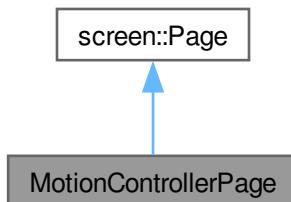
Implements [Feedback](#).

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

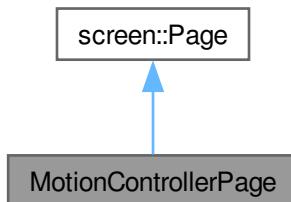
- include/utils/controls/motion\_controller.h
- src/utils/controls/motion\_controller.cpp

## 5.49 MotionControllerPage Class Reference

Inheritance diagram for MotionControllerPage:



Collaboration 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)*

### 5.49.1 Member Function Documentation

#### 5.49.1.1 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

<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](#).

#### 5.49.1.2 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

<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](#).

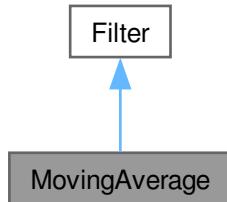
The documentation for this class was generated from the following file:

- src/utils/controls/motion\_controller.cpp

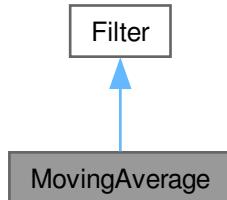
## 5.50 MovingAverage Class Reference

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

Inheritance diagram for MovingAverage:



Collaboration 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](#)

### 5.50.1 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).

## 5.50.2 Constructor & Destructor Documentation

### 5.50.2.1 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
--------------------	---

### 5.50.2.2 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

## 5.50.3 Member Function Documentation

### 5.50.3.1 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

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

Implements [Filter](#).

### 5.50.3.2 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

### 5.50.3.3 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](#).

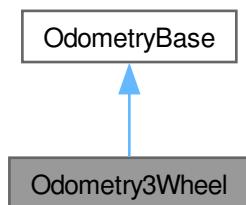
The documentation for this class was generated from the following files:

- include/utils/moving\_average.h
- src/utils/moving\_average.cpp

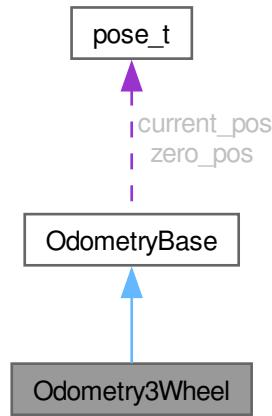
## 5.51 Odometry3Wheel Class Reference

```
#include <odometry_3wheel.h>
```

Inheritance diagram for Odometry3Wheel:



Collaboration 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}

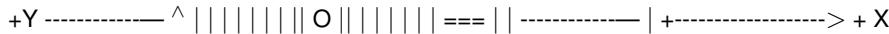
## Protected Attributes inherited from [OdometryBase](#)

- vex::task \* `handle`
- vex::mutex `mut`
- `pose_t current_pos`
- double `speed`
- double `accel`
- double `ang_speed_deg`
- double `ang_accel_deg`

### 5.51.1 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:



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

### 5.51.2 Constructor & Destructor Documentation

#### 5.51.2.1 [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

**5.51.3 Member Function Documentation****5.51.3.1 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

**5.51.3.2 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

## 5.52 Odometry3Wheel::odometry3wheel\_cfg\_t Struct Reference

```
#include <odometry_3wheel.h>
```

### Public Attributes

- double wheelbase\_dist
- double off\_axis\_center\_dist
- double wheel\_diam

### 5.52.1 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

### 5.52.2 Member Data Documentation

#### 5.52.2.1 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

#### 5.52.2.2 wheel\_diam

```
double Odometry3Wheel::odometry3wheel_cfg_t::wheel_diam
```

the diameter of the tracking wheel

#### 5.52.2.3 wheelbase\_dist

```
double Odometry3Wheel::odometry3wheel_cfg_t::wheelbase_dist
```

distance from the center of the left wheel to the center of the right wheel

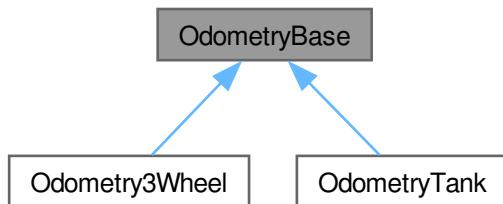
The documentation for this struct was generated from the following file:

- include/subsystems/odometry/odometry\_3wheel.h

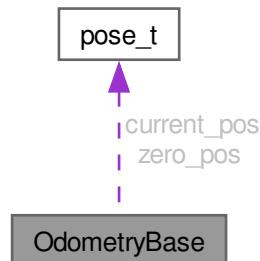
## 5.53 OdometryBase Class Reference

```
#include <odometry_base.h>
```

Inheritance diagram for OdometryBase:



Collaboration 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`

## 5.53.1 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

## 5.53.2 Constructor & Destructor Documentation

### 5.53.2.1 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 <a href="#">update()</a> manually
-----------------	---

### 5.53.3 Member Function Documentation

#### 5.53.3.1 `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 <a href="#">OdometryBase</a> object
------------	--

**Returns**

Required integer return code. Unused.

#### 5.53.3.2 `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.

#### 5.53.3.3 `get_accel()`

```
double OdometryBase::get_accel ( )
```

Get the current acceleration

**Returns**

the acceleration rate of the robot (inch/s<sup>2</sup>)

#### 5.53.3.4 `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/s<sup>2</sup>)

### 5.53.3.5 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)

### 5.53.3.6 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

### 5.53.3.7 get\_speed()

```
double OdometryBase::get_speed ( )
```

Get the current speed

#### Returns

the speed at which the robot is moving and grooving (inch/s)

### 5.53.3.8 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

**5.53.3.9 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

**5.53.3.10 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](#).

**5.53.3.11 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.

**5.53.3.12 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](#).

**5.53.4 Member Data Documentation****5.53.4.1 accel**

```
double OdometryBase::accel [protected]
```

the rate at which we are accelerating (inch/s<sup>2</sup>)

**5.53.4.2 ang\_accel\_deg**

```
double OdometryBase::ang_accel_deg [protected]
```

the rate at which we are accelerating our turn (deg/s<sup>2</sup>)

**5.53.4.3 ang\_speed\_deg**

```
double OdometryBase::ang_speed_deg [protected]
```

the speed at which we are turning (deg/s)

#### 5.53.4.4 current\_pos

```
pose_t OdometryBase::current_pos [protected]
```

Current position of the robot in terms of x,y,rotation

#### 5.53.4.5 handle

```
vex::task* OdometryBase::handle [protected]
```

handle to the vex task that is running the odometry code

#### 5.53.4.6 mut

```
vex::mutex OdometryBase::mut [protected]
```

Mutex to control multithreading

#### 5.53.4.7 speed

```
double OdometryBase::speed [protected]
```

the speed at which we are travelling (inch/s)

#### 5.53.4.8 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:

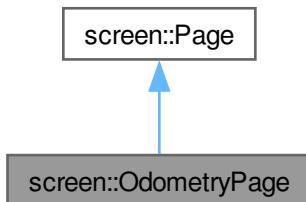
- include/subsystems/odometry/odometry\_base.h
- src/subsystems/odometry/odometry\_base.cpp

## 5.54 screen::OdometryPage Class Reference

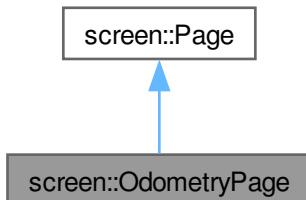
a page that shows odometry position and rotation and a map (if an sd card with the file is on)

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

Inheritance diagram for screen::OdometryPage:



Collaboration 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](#)

### 5.54.1 Detailed Description

a page that shows odometry position and rotation and a map (if an sd card with the file is on)

## 5.54.2 Constructor & Destructor Documentation

### 5.54.2.1 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

## 5.54.3 Member Function Documentation

### 5.54.3.1 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](#).

### 5.54.3.2 update()

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

#### See also

[Page::update](#)

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

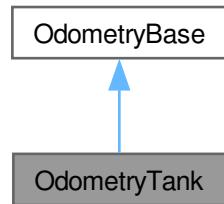
The documentation for this class was generated from the following files:

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

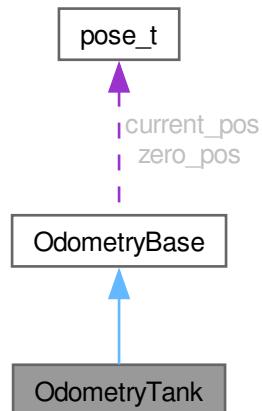
## 5.55 OdometryTank Class Reference

```
#include <odometry_tank.h>
```

Inheritance diagram for OdometryTank:



Collaboration 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](#)

## 5.55.1 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.

## 5.55.2 Constructor & Destructor Documentation

### 5.55.2.1 [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 <a href="#">robot_specs_t</a> 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 <a href="#">update()</a> .

**5.55.2.2 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 <a href="#">robot_specs_t</a> 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 <a href="#">update()</a> .

**5.55.2.3 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 <a href="#">robot_specs_t</a> 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 <a href="#">update()</a> .

### 5.55.3 Member Function Documentation

#### 5.55.3.1 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

<i>newpos</i>	the position the odometry will take
---------------	-------------------------------------

Resets the position and rotational data to the input.

Reimplemented from [OdometryBase](#).

#### 5.55.3.2 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](#).

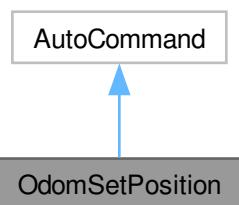
The documentation for this class was generated from the following files:

- include/subsystems/odometry/odometry\_tank.h
- src/subsystems/odometry/odometry\_tank.cpp

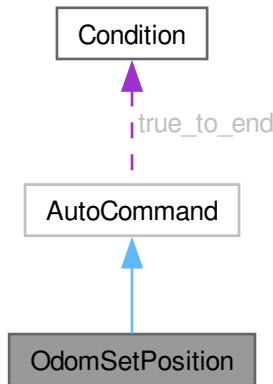
## 5.56 OdomSetPosition Class Reference

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

Inheritance diagram for OdomSetPosition:



Collaboration 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`

#### 5.56.1 Detailed Description

[AutoCommand](#) wrapper class for the `set_position` function in the `Odometry` class

## 5.56.2 Constructor & Destructor Documentation

### 5.56.2.1 OdomSetPosition()

```
OdomSetPosition::OdomSetPosition (
    OdometryBase & odom,
    const pose_t & newpos = OdometryBase::zero_pos )
```

constructs a new [OdomSetPosition](#) command

**Parameters**

<i>odom</i>	the odometry system we are setting
<i>newpos</i>	the position we are telling the odometry to take. defaults to (0, 0), angle = 90

Construct an Odometry set pos

**Parameters**

<i>odom</i>	the odometry system we are setting
<i>newpos</i>	the now position to set the odometry to

### 5.56.3 Member Function Documentation

#### 5.56.3.1 run()

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

Run set\_position Overrides run from [AutoCommand](#)

**Returns**

true when execution is complete, false otherwise

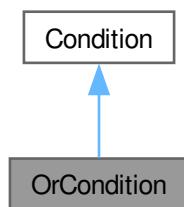
Reimplemented from [AutoCommand](#).

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

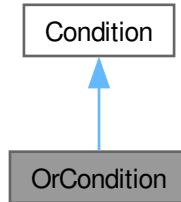
- include/utils/command\_structure/drive\_commands.h
- src/utils/command\_structure/drive\_commands.cpp

## 5.57 OrCondition Class Reference

Inheritance diagram for OrCondition:



Collaboration 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)

### 5.57.1 Member Function Documentation

#### 5.57.1.1 [test\(\)](#)

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

Implements [Condition](#).

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

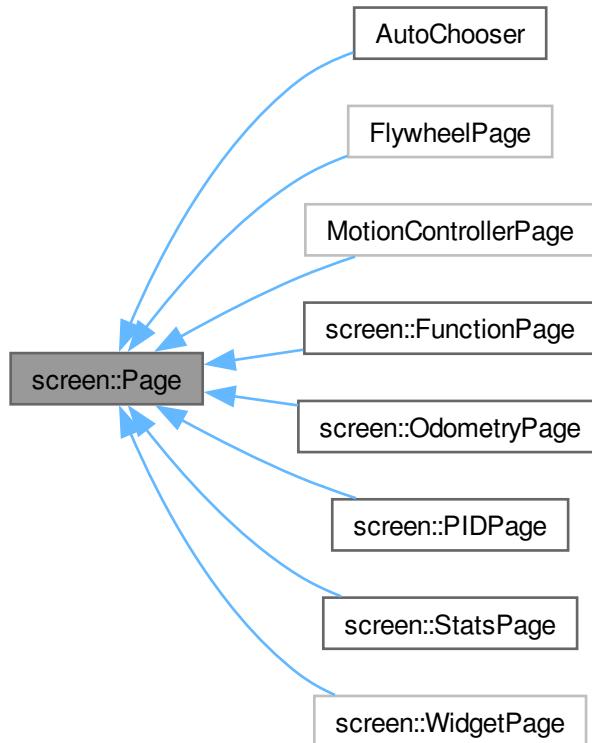
- src/utils/command\_structure/auto\_command.cpp

## 5.58 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)*

### 5.58.1 Detailed Description

[Page](#) describes one part of the screen slideshow.

### 5.58.2 Member Function Documentation

#### 5.58.2.1 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](#), and [FlywheelPage](#).

**5.58.2.2 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](#), [screen::WidgetPage](#), [screen::StatsPage](#), [screen::OdometryPage](#), [screen::FunctionPage](#), [screen::PIDPage](#), [MotionControllerPage](#), and [FlywheelPage](#).

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

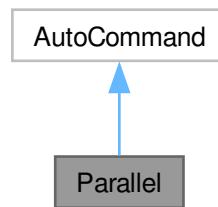
- include/subsystems/screen.h

## 5.59 Parallel Class Reference

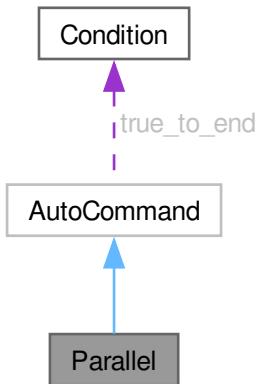
**Parallel** runs multiple commands in parallel and waits for all to finish before continuing. if none finish before this command's timeout, it will call `on_timeout` on all children continue.

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

Inheritance diagram for Parallel:



Collaboration 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

### 5.59.1 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.

## 5.59.2 Member Function Documentation

### 5.59.2.1 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](#).

### 5.59.2.2 run()

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

Executes the command Overridden by child classes

#### Returns

true when the command is finished, false otherwise

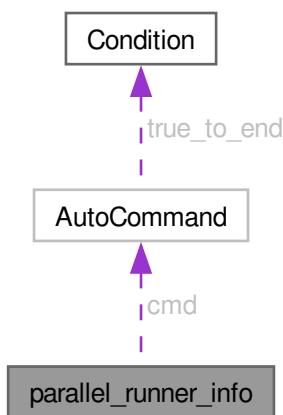
Reimplemented from [AutoCommand](#).

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

## 5.60 parallel\_runner\_info Struct Reference

Collaboration diagram for parallel\_runner\_info:



## 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

## 5.61 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** ()

### 5.61.1 Detailed Description

Wrapper for a vector of points, checking if any of the points are too close for pure pursuit

### 5.61.2 Constructor & Destructor Documentation

#### 5.61.2.1 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

### 5.61.3 Member Function Documentation

#### 5.61.3.1 get\_points()

```
std::vector< point_t > PurePursuit::Path::get_points ( )
```

Get the points associated with this Path

#### 5.61.3.2 get\_radius()

```
double PurePursuit::Path::get_radius ( )
```

Get the radius associated with this Path

#### 5.61.3.3 is\_valid()

```
bool PurePursuit::Path::is_valid ( )
```

Get whether this path will behave as expected

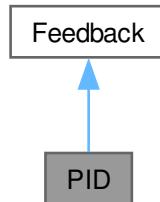
The documentation for this class was generated from the following files:

- include/utils/pure\_pursuit.h
- src/utils/pure\_pursuit.cpp

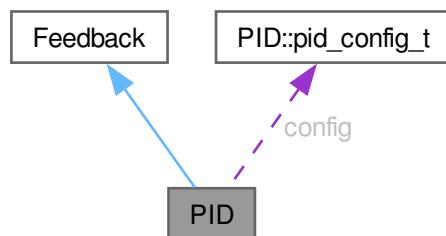
## 5.62 PID Class Reference

```
#include <pid.h>
```

Inheritance diagram for PID:



Collaboration 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](#)

**5.62.1 Detailed Description****PID Class**

Defines a standard feedback loop using the constants kP, kI, kD, deadband, and on\_target\_time. The formula is:

$\text{out} = \text{kP} * \text{error} + \text{kI} * \text{integral}(\text{d Error}) + \text{kD} * (\text{dError}/\text{dt})$

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

**5.62.2 Member Enumeration Documentation****5.62.2.1 ERROR\_TYPE**

enum [PID::ERROR\\_TYPE](#)

An enum to distinguish between a linear and angular calculation of [PID](#) error.

**5.62.3 Constructor & Destructor Documentation****5.62.3.1 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

## 5.62.4 Member Function Documentation

### 5.62.4.1 [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](#).

### 5.62.4.2 [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

### 5.62.4.3 [get\\_sensor\\_val\(\)](#)

```
double PID::get_sensor_val ( ) const
```

gets the sensor value that we were last updated with

**Returns**

`sensor_val`

#### 5.62.4.4 get\_target()

```
double PID::get_target ( ) const
```

Get the [PID](#)'s target

##### Returns

the target the [PID](#) controller is trying to achieve

#### 5.62.4.5 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 <a href="#">PID</a> . necessary to satisfy <a href="#">Feedback</a> base
<code>set_pt</code>	sets the target of the <a href="#">PID</a> controller
<code>start_vel</code>	completely ignored for <a href="#">PID</a> . necessary to satisfy <a href="#">Feedback</a> base
<code>end_vel</code>	sets the target end velocity of the <a href="#">PID</a> controller

Implements [Feedback](#).

#### 5.62.4.6 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](#).

#### 5.62.4.7 reset()

```
void PID::reset ( )
```

Reset the [PID](#) loop by resetting time since 0 and accumulated error.

#### 5.62.4.8 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 <a href="#">PID</a> controller will never command the output go below <i>lower</i>
<i>upper</i>	the upper limit. the <a href="#">PID</a> 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](#).

#### 5.62.4.9 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

#### 5.62.4.10 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](#).

## 5.62.5 Member Data Documentation

### 5.62.5.1 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`

## 5.63 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**

### 5.63.1 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

## 5.63.2 Member Data Documentation

### 5.63.2.1 error\_method

`ERROR_TYPE PID::pid_config_t::error_method`

Linear or angular. wheter to do error as a simple subtraction or to wrap

### 5.63.2.2 on\_target\_time

```
double PID::pid_config_t::on_target_time
```

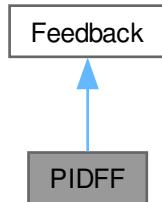
the time in seconds that we have to be on target for to say we are officially at the target

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

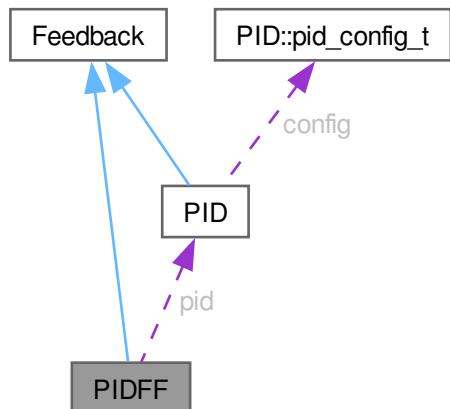
- include/utils/controls/pid.h

## 5.64 PIDFF Class Reference

Inheritance diagram for PIDFF:



Collaboration 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**

### 5.64.1 Member Function Documentation

#### 5.64.1.1 get()

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

##### Returns

the last saved result from the feedback controller

Implements [Feedback](#).

#### 5.64.1.2 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

<code>start_pt</code>	the current sensor value
<code>set_pt</code>	where the sensor value should be
<code>start_vel</code>	the current rate of change of the sensor value
<code>end_vel</code>	the desired ending rate of change of the sensor value

Initialize the feedback controller for a movement

**Parameters**

<i>start</i> <i>_pt</i>	the current sensor value
<i>set_pt</i>	where the sensor value should be

Implements [Feedback](#).

**5.64.1.3 is\_on\_target()**

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

**Returns**

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

**5.64.1.4 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](#).

**5.64.1.5 set\_target()**

```
void PIDFF::set_target (
    double set_pt )
```

Set the target of the [PID](#) loop

**Parameters**

<i>set</i> <i>_pt</i>	Setpoint / target value
--------------------------	-------------------------

### 5.64.1.6 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](#).

### 5.64.1.7 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:

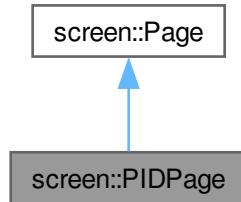
- include/utils/controls/pidff.h
- src/utils/controls/pidff.cpp

## 5.65 screen::PIDPage Class Reference

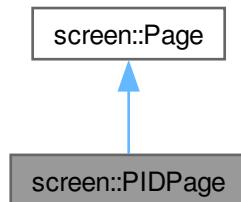
[PIDPage](#) provides a way to tune a pid controller on the screen.

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

Inheritance diagram for screen::PIDPage:



Collaboration 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

## 5.65.1 Detailed Description

[PIDPage](#) provides a way to tune a pid controller on the screen.

## 5.65.2 Constructor & Destructor Documentation

### 5.65.2.1 PIDPage()

```
screen::PIDPage::PIDPage (
    PID & pid,
    std::string name,
    std::function< void(void)> onchange = []() {} )
```

Create a [PIDPage](#).

**Parameters**

<i>pid</i>	the pid controller we're changing
<i>name</i>	a name to recognize this pid controller if we've got multiple pid screens
<i>onchange</i>	a function that is called when a tuning parameter is changed. If you need to update stuff on that change register a handler here

**5.65.3 Member Function Documentation****5.65.3.1 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](#).

**5.65.3.2 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

**5.66 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*

### 5.66.1 Detailed Description

Data structure representing an X,Y coordinate

### 5.66.2 Member Function Documentation

#### 5.66.2.1 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

#### 5.66.2.2 operator+()

```
point_t point_t::operator+ (
    const point_t & other ) const [inline]
```

Vector2D addition operation on points

**Parameters**

<i>other</i>	the point to add on to this
--------------	-----------------------------

**Returns**

this + other (this.x + other.x, this.y + other.y)

**5.66.2.3 operator-()**

```
point_t point_t::operator- (
    const point_t & other ) const [inline]
```

Vector2D subtraction operation on points

**Parameters**

<i>other</i>	the <a href="#">point_t</a> 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](#)

**5.67 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*

### 5.67.1 Detailed Description

Describes a single position and rotation

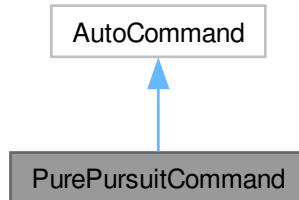
The documentation for this struct was generated from the following file:

- include/utils/geometry.h

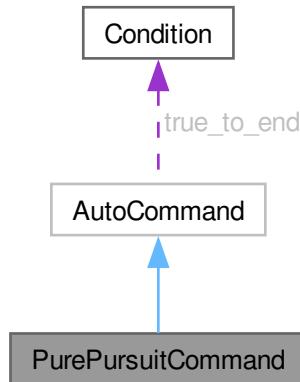
## 5.68 PurePursuitCommand Class Reference

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

Inheritance diagram for PurePursuitCommand:



Collaboration 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`

### 5.68.1 Detailed Description

Autocommand wrapper class for pure pursuit function in the [TankDrive](#) class

### 5.68.2 Constructor & Destructor Documentation

#### 5.68.2.1 `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

<code>path</code>	The list of coordinates to follow, in order
<code>dir</code>	Run the bot forwards or backwards
<code>feedback</code>	The feedback controller determining speed
<code>max_speed</code>	Limit the speed of the robot (for pid / pidff feedbacks)

### 5.68.3 Member Function Documentation

#### 5.68.3.1 on\_timeout()

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

Reset the drive system when it times out

Reimplemented from [AutoCommand](#).

#### 5.68.3.2 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](#)

## 5.69 Rect Struct Reference

Collaboration diagram for Rect:



### 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

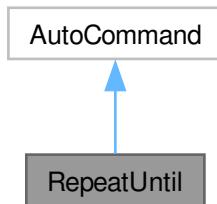
- `point_t min`
- `point_t max`

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

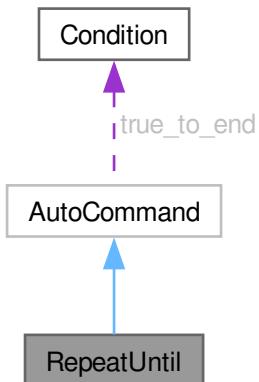
- `include/utils/geometry.h`

## 5.70 RepeatUntil Class Reference

Inheritance diagram for RepeatUntil:



Collaboration 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`

## 5.70.1 Constructor & Destructor Documentation

### 5.70.1.1 `RepeatUntil()` [1/2]

```
RepeatUntil::RepeatUntil (
    InOrder cmd,
    size_t repeats )
```

`RepeatUntil` that runs a fixed number of times.

#### Parameters

<code>cmds</code>	the cmd to repeat
<code>repeats</code>	the number of repeats to do

### 5.70.1.2 `RepeatUntil()` [2/2]

```
RepeatUntil::RepeatUntil (
    InOrder cmd,
    Condition * true_to_end )
```

`RepeatUntil` the condition.

**Parameters**

<i>cmds</i>	the cmd's to run
<i>true_to_end</i>	we will repeat until <i>true_or_end.test()</i> returns true

**5.70.2 Member Function Documentation****5.70.2.1 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](#).

**5.70.2.2 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](#).

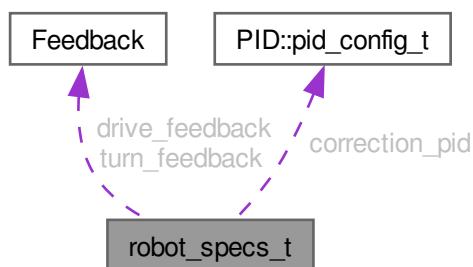
The documentation for this class was generated from the following files:

- include/utils/command\_structure/auto\_command.h
- src/utils/command\_structure/auto\_command.cpp

**5.71 robot\_specs\_t Struct Reference**

```
#include <robot_specs.h>
```

Collaboration diagram for robot\_specs\_t:



**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 default 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*

**5.71.1 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

**5.72 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**

### 5.72.1 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`

## 5.73 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`

## 5.74 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*

### 5.74.1 Detailed Description

Serializes Arbitrary data to a file on the SD Card.

### 5.74.2 Constructor & Destructor Documentation

#### 5.74.2.1 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>

### 5.74.3 Member Function Documentation

#### 5.74.3.1 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

#### 5.74.3.2 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

**5.74.3.3 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

**5.74.3.4 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

**5.74.3.5 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

### 5.74.3.6 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

### 5.74.3.7 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

### 5.74.3.8 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

### 5.74.3.9 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**

<code>name</code>	name of value
<code>otherwise</code>	value if the name is not specified

**Returns**

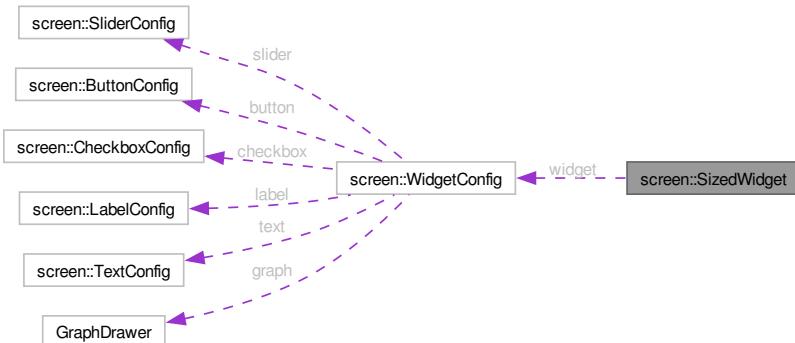
the value if found or otherwise

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

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

## 5.75 screen::SizedWidget Struct Reference

Collaboration diagram for screen::SizedWidget:

**Public Attributes**

- int **size**
- [WidgetConfig](#) & **widget**

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

- include/subsystems/screen.h

## 5.76 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

## 5.77 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

## 5.78 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*

### 5.78.1 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.

## 5.78.2 Constructor & Destructor Documentation

### 5.78.2.1 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

**5.78.3 Member Function Documentation****5.78.3.1 update()**

```
bool screen::SliderWidget::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 value updated

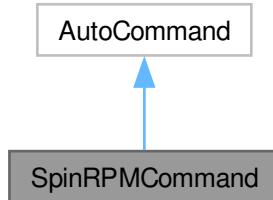
The documentation for this class was generated from the following files:

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

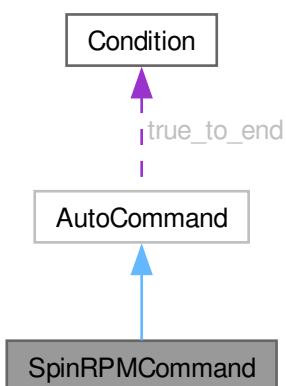
**5.79 SpinRPMCommand Class Reference**

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

Inheritance diagram for SpinRPMCommand:



Collaboration 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

### 5.79.1 Detailed Description

File: [flywheel\\_commands.h](#) Desc: [insert meaningful desc] [AutoCommand](#) wrapper class for the spin\_rpm function in the [Flywheel](#) class

### 5.79.2 Constructor & Destructor Documentation

#### 5.79.2.1 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]

### 5.79.3 Member Function Documentation

#### 5.79.3.1 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

## 5.80 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**

### 5.80.1 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:

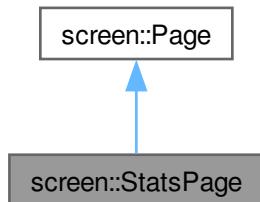
- include/utils/pure\_pursuit.h

## 5.81 screen::StatsPage Class Reference

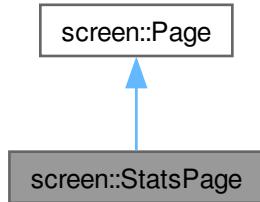
Draws motor stats and battery stats to the screen.

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

Inheritance diagram for screen::StatsPage:



Collaboration 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

### 5.81.1 Detailed Description

Draws motor stats and battery stats to the screen.

### 5.81.2 Constructor & Destructor Documentation

#### 5.81.2.1 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
---------------	--

### 5.81.3 Member Function Documentation

#### 5.81.3.1 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](#).

#### 5.81.3.2 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:

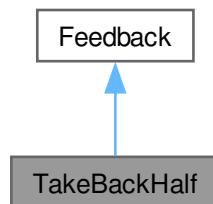
- include/subsystems/screen.h
- src/subsystems/screen.cpp

## 5.82 TakeBackHalf Class Reference

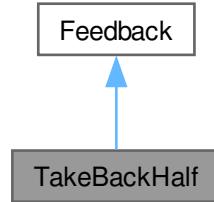
A velocity controller.

```
#include <take_back_half.h>
```

Inheritance diagram for TakeBackHalf:



Collaboration 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**

### 5.82.1 Detailed Description

A velocity controller.

#### Warning

If you try to use this as a position controller, it will fail.

### 5.82.2 Member Function Documentation

#### 5.82.2.1 `get()`

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

#### Returns

the last saved result from the feedback controller

Implements [Feedback](#).

### 5.82.2.2 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](#).

### 5.82.2.3 is\_on\_target()

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

#### Returns

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

### 5.82.2.4 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](#).

### 5.82.2.5 update()

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

Iterate the feedback loop once with an updated sensor value

**Parameters**

<code>val</code>	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

## 5.83 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 \\* 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`)
- `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)

### 5.83.1 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

### 5.83.2 Member Enumeration Documentation

#### 5.83.2.1 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

### 5.83.3 Constructor & Destructor Documentation

#### 5.83.3.1 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 <a href="#">robot_specs_t</a> for more info
<i>odom</i>	an odometry system to track position and rotation. this is necessary to execute autonomous paths

**5.83.4 Member Function Documentation****5.83.4.1 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

**5.83.4.2 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

### 5.83.4.3 `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

#### 5.83.4.4 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

#### 5.83.4.5 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)

#### 5.83.4.6 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
----------	------------------------------

## Parameters

<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

**5.83.4.7 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

**5.83.4.8 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

**5.83.4.9 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

### 5.83.4.10 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

**5.83.4.11 reset\_auto()**

```
void TankDrive::reset_auto ( )
```

Reset the initialization for autonomous drive functions

**5.83.4.12 stop()**

```
void TankDrive::stop ( )
```

Stops rotation of all the motors using their "brake mode"

**5.83.4.13 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

### 5.83.4.14 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

#### 5.83.4.15 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 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

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

#### 5.83.4.16 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

## 5.84 screen::TextConfig Struct Reference

**Public Attributes**

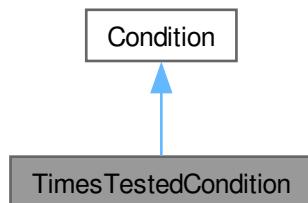
- std::function< std::string()> **text**

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

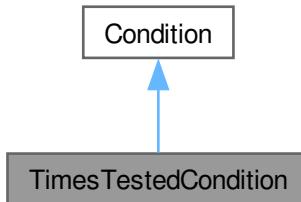
- include/subsystems/screen.h

## 5.85 TimesTestedCondition Class Reference

Inheritance diagram for TimesTestedCondition:



Collaboration 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)

## 5.85.1 Member Function Documentation

### 5.85.1.1 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

## 5.86 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*

### 5.86.1 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

## 5.87 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

### 5.87.1 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

## 5.87.2 Constructor & Destructor Documentation

### 5.87.2.1 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

## 5.87.3 Member Function Documentation

### 5.87.3.1 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</i> ← _s	Time since start of movement
<i>pos</i> ← _s	The current position

**Returns**

[motion\\_t](#) Position, velocity and acceleration

**5.87.3.2 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</i> ← _s	Time since start of movement
---------------------	------------------------------

**Returns**

[motion\\_t](#) Position, velocity and acceleration

**5.87.3.3 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

**5.87.3.4 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**

<i>accel</i>	the acceleration amount to use
--------------	--------------------------------

### 5.87.3.5 set\_endpts()

```
void TrapezoidProfile::set_endpts (
    double start,
    double end )
```

set\_endpts defines a start and end position

#### Parameters

<i>start</i>	the starting position of the path
<i>end</i>	the ending position of the path

### 5.87.3.6 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

<i>max_v</i>	the maximum velocity the robot can travel at
--------------	--

### 5.87.3.7 set\_vel\_endpts()

```
void TrapezoidProfile::set_vel_endpts (
    double start,
    double end )
```

set start and end velocities

#### Parameters

<i>start</i>	the starting velocity of the path
<i>end</i>	the ending velocity of the path

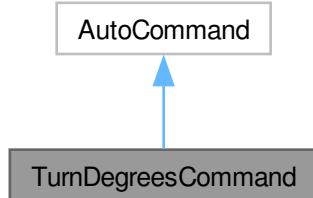
The documentation for this class was generated from the following files:

- include/utils/controls/trapezoid\_profile.h
- src/utils/trapezoid\_profile.cpp

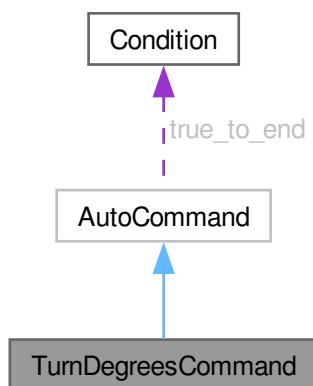
## 5.88 TurnDegreesCommand Class Reference

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

Inheritance diagram for TurnDegreesCommand:



Collaboration 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

## 5.88.1 Detailed Description

[AutoCommand](#) wrapper class for the turn\_degrees function in the [TankDrive](#) class

## 5.88.2 Constructor & Destructor Documentation

### 5.88.2.1 TurnDegreesCommand()

```
TurnDegreesCommand::TurnDegreesCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    double degrees,
    double max_speed = 1,
    double end_speed = 0 )
```

Construct a [TurnDegreesCommand](#) Command

#### Parameters

<code>drive_sys</code>	the drive system we are commanding
<code>feedback</code>	the feedback controller we are using to execute the turn
<code>degrees</code>	how many degrees to rotate
<code>max_speed</code>	0 -> 1 percentage of the drive systems speed to drive at

## 5.88.3 Member Function Documentation

### 5.88.3.1 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](#).

### 5.88.3.2 run()

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

Run turn\_degrees Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

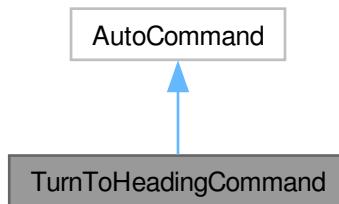
The documentation for this class was generated from the following files:

- include/utils/command\_structure/drive\_commands.h
- src/utils/command\_structure/drive\_commands.cpp

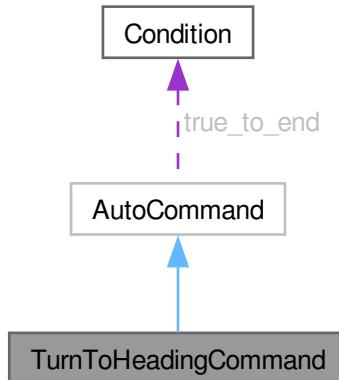
## 5.89 TurnToHeadingCommand Class Reference

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

Inheritance diagram for TurnToHeadingCommand:



Collaboration 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`

## 5.89.1 Detailed Description

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

## 5.89.2 Constructor & Destructor Documentation

### 5.89.2.1 TurnToHeadingCommand()

```
TurnToHeadingCommand::TurnToHeadingCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    double heading_deg,
    double max_speed = 1,
    double end_speed = 0 )
```

Construct a [TurnToHeadingCommand](#) Command

#### Parameters

<code>drive_sys</code>	the drive system we are commanding
<code>feedback</code>	the feedback controller we are using to execute the drive
<code>heading_deg</code>	the heading to turn to in degrees
<code>max_speed</code>	0 -> 1 percentage of the drive systems speed to drive at

### 5.89.3 Member Function Documentation

#### 5.89.3.1 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](#).

#### 5.89.3.2 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

## 5.90 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- \(const Vector2D &other\)](#)

### 5.90.1 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

### 5.90.2 Constructor & Destructor Documentation

#### 5.90.2.1 `Vector2D()` [1/2]

```
Vector2D::Vector2D (
    double dir,
    double mag )
```

Construct a vector object.

**Parameters**

<i>dir</i>	Direction, in radians. 'forward' is 0, clockwise positive when viewed from the top.
<i>mag</i>	Magnitude.

**5.90.2.2 Vector2D() [2/2]**

```
Vector2D::Vector2D (
    point_t p )
```

Construct a vector object from a cartesian point.

**Parameters**

<i>p</i>	point_t.x , point_t.y
----------	-----------------------

**5.90.3 Member Function Documentation****5.90.3.1 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.

**5.90.3.2 get\_mag()**

```
double Vector2D::get_mag ( ) const
```

**Returns**

the magnitude of the vector

Get the magnitude of the vector

### 5.90.3.3 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.

### 5.90.3.4 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.

### 5.90.3.5 normalize()

```
Vector2D Vector2D::normalize ( )
```

Changes the magnitude of the vector to 1

#### Returns

the normalized vector

Changes the magnetude of the vector to 1

### 5.90.3.6 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

### 5.90.3.7 operator+( )

```
Vector2D Vector2D::operator+ (
    const Vector2D & other )
```

Add the components of two vectors together `Vector2D + Vector2D = (this.x + other.x, this.y + other.y)`

#### Parameters

<i>other</i>	the vector to add to this
--------------	---------------------------

#### Returns

the sum of the vectors

### 5.90.3.8 operator-( )

```
Vector2D Vector2D::operator- (
    const Vector2D & other )
```

Subtract the components of two vectors together `Vector2D - Vector2D = (this.x - other.x, this.y - other.y)`

#### Parameters

<i>other</i>	the vector to subtract from this
--------------	----------------------------------

#### Returns

the difference of the vectors

### 5.90.3.9 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:

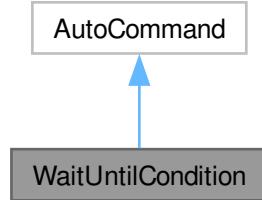
- include/utils/vector2d.h
- src/utils/vector2d.cpp

## 5.91 WaitUntilCondition Class Reference

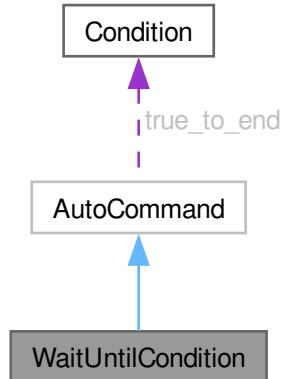
Waits until the condition is true.

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

Inheritance diagram for WaitUntilCondition:



Collaboration 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

## 5.91.1 Detailed Description

Waits until the condition is true.

## 5.91.2 Member Function Documentation

### 5.91.2.1 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](#).

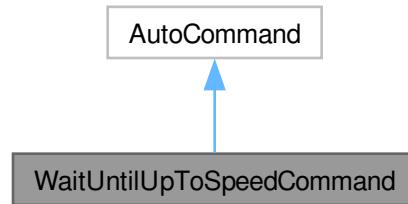
The documentation for this class was generated from the following file:

- include/utils/command\_structure/auto\_command.h

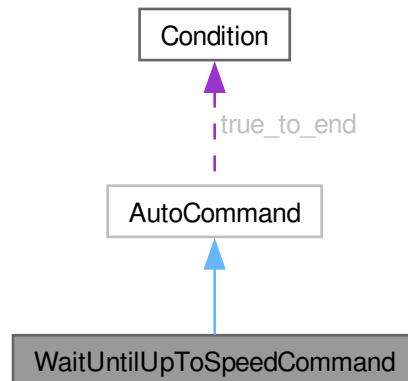
## 5.92 WaitUntilUpToSpeedCommand Class Reference

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

Inheritance diagram for WaitUntilUpToSpeedCommand:



Collaboration 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

## 5.92.1 Detailed Description

[AutoCommand](#) that listens to the [Flywheel](#) and waits until it is at its target speed +/- the specified threshold

## 5.92.2 Constructor & Destructor Documentation

### 5.92.2.1 [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

## 5.92.3 Member Function Documentation

### 5.92.3.1 [run\(\)](#)

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

Run spin\_manual Overrides run from [AutoCommand](#)

#### Returns

true when execution is complete, false otherwise

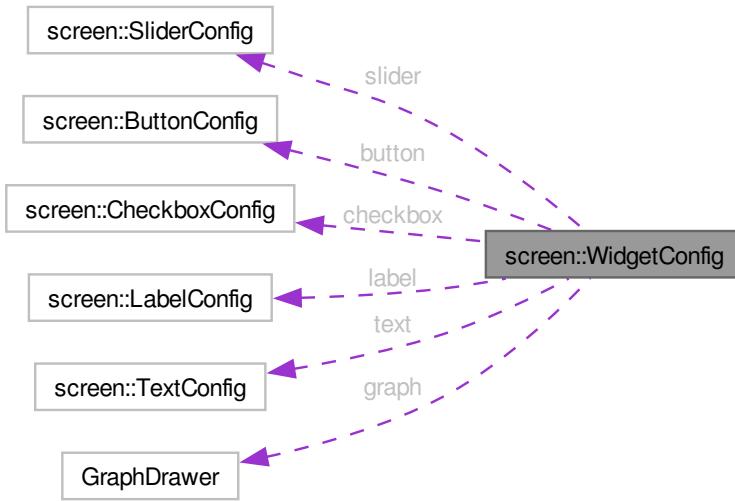
Reimplemented from [AutoCommand](#).

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

## 5.93 screen::WidgetConfig Struct Reference

Collaboration diagram for screen::WidgetConfig:



### Public Types

- enum **Type** {
 **Col** , **Row** , **Slider** , **Button** ,
 **Checkbox** , **Label** , **Text** , **Graph** }

### Public Attributes

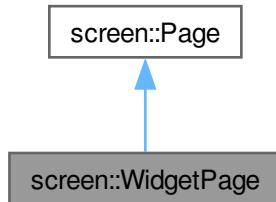
- Type **type**
- union {
 std::vector< [SizedWidget](#) > **widgets**
**SliderConfig** **slider**
**ButtonConfig** **button**
**CheckboxConfig** **checkbox**
**LabelConfig** **label**
**TextConfig** **text**
**GraphDrawer** \* **graph**
} **config**

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

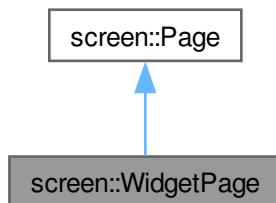
- include/subsystems/screen.h

## 5.94 screen::WidgetPage Class Reference

Inheritance diagram for screen::WidgetPage:



Collaboration 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)*

### 5.94.1 Member Function Documentation

#### 5.94.1.1 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](#).

**5.94.1.2 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 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
```

## 6.5 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
00010 using namespace vex;
00011 using namespace std;
00012
00020 template <typename T>
00021 class Lift
00022 {
00023     public:
00024
00031     struct lift_cfg_t
00032     {
00033         double up_speed, down_speed;
00034         double softstop_up, softstop_down;
00035
00036         PID::pid_config_t lift_pid_cfg;
00037     };
00038
00060     Lift(motor_group &lift_motors, lift_cfg_t &lift_cfg, map<T, double> &setpoint_map, limit
00061     *homing_switch=NULL)
00061     : lift_motors(lift_motors), cfg(lift_cfg), lift_pid(cfg.lift_pid_cfg), setpoint_map(setpoint_map),
00062     homing_switch(homing_switch)
00062     {
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
00093     void control_continuous(bool up_ctrl, bool down_ctrl)
00094     {
00095         static timer tmr;
00096
00097         double cur_pos = 0;
00098
00099         // Check if there's a hook for a custom sensor. If not, use the motors.
00100         if(get_sensor == NULL)
00101             cur_pos = lift_motors.position(rev);
00102         else
00103             cur_pos = get_sensor();
00104
00105         if(up_ctrl && cur_pos < cfg.softstop_up)
00106         {
00107             lift_motors.spin(directionType::fwd, cfg.up_speed, volt);
00108             setpoint = cur_pos + .3;
00109
00110             // std::cout << "DEBUG OUT: UP " << setpoint << ", " << tmr.time(sec) << ", " << cfg.down_speed <
00111             "\n";
00112
00113             // Disable the PID while going UP.
00114             is_async = false;
00114         } else if(down_ctrl && cur_pos > cfg.softstop_down)
00115         {
```

```

00116     // Lower the lift slowly, at a rate defined by down_speed
00117     if(setpoint > cfg.softstop_down)
00118         setpoint = setpoint - (tmr.time(sec) * cfg.down_speed);
00119     // std::cout << "DEBUG OUT: DOWN " << setpoint << ", " << tmr.time(sec) << ", " << cfg.down_speed <<
00120     "\n";
00121     is_async = true;
00122 } else
00123 {
00124     // Hold the lift at the last setpoint
00125     is_async = true;
00126 }
00127 tmr.reset();
00128 }
00129
00138 void control_manual(bool up_btn, bool down_btn, int volt_up, int volt_down)
00139 {
00140     static bool down_hold = false;
00141     static bool init = true;
00142
00143     // Allow for setting position while still calling this function
00144     if(init || up_btn || down_btn)
00145     {
00146         init = false;
00147         is_async = false;
00148     }
00149
00150     double rev = lift_motors.position(rotationUnits::rev);
00151
00152     if(rev < cfg.softstop_down && down_btn)
00153         down_hold = true;
00154     else if( !down_btn )
00155         down_hold = false;
00156
00157     if(up_btn && rev < cfg.softstop_up)
00158         lift_motors.spin(directionType::fwd, volt_up, voltageUnits::volt);
00159     else if(down_btn && rev > cfg.softstop_down && !down_hold)
00160         lift_motors.spin(directionType::rev, volt_down, voltageUnits::volt);
00161     else
00162         lift_motors.spin(directionType::fwd, 0, voltageUnits::volt);
00163 }
00164
00165 void control_setpoints(bool up_step, bool down_step, vector<T> pos_list)
00166 {
00167     // Make sure inputs are only processed on the rising edge of the button
00168     static bool up_last = up_step, down_last = down_step;
00169
00170     bool up_rising = up_step && !up_last;
00171     bool down_rising = down_step && !down_last;
00172
00173     up_last = up_step;
00174     down_last = down_step;
00175
00176     static int cur_index = 0;
00177
00178     // Avoid an index overflow. Shouldn't happen unless the user changes pos_list between calls.
00179     if(cur_index >= pos_list.size())
00180         cur_index = pos_list.size() - 1;
00181
00182     // Increment or decrement the index of the list, bringing it up or down.
00183     if(up_rising && cur_index < (pos_list.size() - 1))
00184         cur_index++;
00185     else if(down_rising && cur_index > 0)
00186         cur_index--;
00187
00188     // Set the lift to hold the position in the background with the PID loop
00189     set_position(pos_list[cur_index]);
00190     is_async = true;
00191 }
00192
00193
00194 bool set_position(T pos)
00195 {
00196     this->setpoint = setpoint_map[pos];
00197     is_async = true;
00198
00199     return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
00200 }
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
0
```

```

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
00291  bool get_async()
00292  {
00293      return is_async;
00294  }
00295
00301  void set_async(bool val)
00302  {
00303      this->is_async = val;
00304  }
00305
00315  void set_sensor_function(double (*fn_ptr) (void) )
00316  {
00317      this->get_sensor = fn_ptr;
00318  }
00319
00326  void set_sensor_reset(void (*fn_ptr) (void) )
00327  {
00328      this->reset_sensor = fn_ptr;
00329  }
00330
00331  private:
00332
00333  motor_group &lift_motors;
00334  lift_cfg_t &cfg;
00335  PID lift_pid;
00336  map<T, double> &setpoint_map;
00337  limit *homing_switch;
00338
00339  atomic<double> setpoint;
00340  atomic<bool> is_async;
00341
00342  double (*get_sensor) (void) = NULL;
00343  void (*reset_sensor) (void) = NULL;
00344
00345
00346 };

```

## 6.6 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
00014 class MecanumDrive
00015 {
00016     public:
00017     struct mecanumdrive_config_t
00018     {
00019         // PID configurations for autonomous driving
00020         PID::pid_config_t drive_pid_conf;
00021         PID::pid_config_t drive_gyro_pid_conf;
00022         PID::pid_config_t turn_pid_conf;
00023
00024         // Diameter of the mecanum wheels
00025         double drive_wheel_diam;
00026
00027         // Diameter of the perpendicular undriven encoder wheel
00028         double lateral_wheel_diam;
00029
00030         // Width between the center of the left and right wheels
00031         double wheelbase_width;
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     bool auto_turn(double degrees, double speed, bool ignore_imu=false);
00046
00047     private:
00048     vex::motor &left_front, &right_front, &left_rear, &right_rear;
00049     mecanumdrive_config_t *config;
00050     vex::rotation *lateral_wheel;
00051     vex::inertial *imu;
00052
00053     PID *drive_pid = NULL;
00054     PID *drive_gyro_pid = NULL;
00055     PID *turn_pid = NULL;
00056
00057     bool init = true;
00058
00059 };

```

## 6.7 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;
00045     } odometry3wheel_cfg_t;
00046
00057     Odometry3Wheel(CustomEncoder &lside_fwd, CustomEncoder &rside_fwd, CustomEncoder &off_axis,
00058     odometry3wheel_cfg_t &cfg, bool is_async=true);
00059
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);
00093     CustomEncoder &lside_fwd, &rside_fwd, &off_axis;
00094     odometry3wheel_cfg_t &cfg;
00095
00096
00097 };

```

## 6.8 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 };

```

## 6.9 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,
00032     vex::inertial *imu=NULL, bool is_async=true);
00042     OdometryTank(CustomEncoder &left_custom_enc, CustomEncoder &right_custom_enc, robot_specs_t
00043     &config, vex::inertial *imu=NULL, bool is_async=true);
00053     OdometryTank(vex::encoder &left_vex_enc, vex::encoder &right_vex_enc, robot_specs_t &config,
00054     vex::inertial *imu=NULL, bool is_async=true);
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,
00074     double rside_diff, double angle_deg);
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 };

```

## 6.10 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"
00010 #include "../core/include/utils/controls/pidff.h"
00011
00012 namespace screen
00013 {
00015     class ButtonWidget
00016     {
00017         public:
00022         ButtonWidget(std::function<void(void)> onpress, Rect rect, std::string name) :
00023             onpress(onpress), rect(rect), name(name) {}
00027         ButtonWidget(void (*onpress)(), Rect rect, std::string name) : onpress(onpress), rect(rect),
00028             name(name) {}
00029
00034         bool update(bool was_pressed, int x, int y);
00036         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
00037
00038     private:
00039         std::function<void(void)> onpress;
00040         Rect rect;
00041         std::string name = "";
00042         bool was_pressed_last = false;
00043     };
00044
00046     class SliderWidget
00047     {
00048         public:
00055         SliderWidget(double &val, double low, double high, Rect rect, std::string name) : value(val),
00056             low(low), high(high), rect(rect), name(name) {}
00056         bool update(bool was_pressed, int x, int y);

```

```
00064     void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
00065
00066     private:
00067         double &value;
00068
00069         double low;
00070         double high;
00071
00072         Rect rect;
00073         std::string name = "";
00074     };
00075
00076     struct WidgetConfig;
00077
00078     struct SliderConfig
00079     {
00080         double &val;
00081         double low;
00082         double high;
00083     };
00084     struct ButtonConfig
00085     {
00086         std::function<void()> onclick;
00087     };
00088     struct CheckboxConfig
00089     {
00090         std::function<void(bool)> onupdate;
00091     };
00092     struct LabelConfig
00093     {
00094         std::string label;
00095     };
00096
00097     struct TextConfig
00098     {
00099         std::function<std::string()> text;
00100     };
00101     struct SizedWidget
00102     {
00103         int size;
00104         WidgetConfig &widget;
00105     };
00106     struct WidgetConfig
00107     {
00108         enum Type
00109         {
00110             Col,
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
```

```

00167  {
00168  public:
00169      WidgetPage(WidgetConfig &cfg) : base_widget(cfg) {}
00170      void update(bool was_pressed, int x, int y) override;
00171
00172      void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override
00173      {
00174          draw_widget(base_widget, {.x1 = 20, .y1 = 0, .x2 = 440, .y2 = 240});
00175      }
00176
00177  private:
00178      WidgetConfig &base_widget;
00179  };
00180
00181  void start_screen(vex::brain::lcd &screen, std::vector<Page *> pages, int first_page = 0);
00182
00183
00184  void next_page();
00185  void prev_page();
00186
00187  void stop_screen();
00188
00189  using update_func_t = std::function<void(bool, int, int)>;
00190
00191  using draw_func_t = std::function<void(vex::brain::lcd &screen, bool, unsigned int)>;
00192
00193  class StatsPage : public Page
00194  {
00195  public:
00196      StatsPage(std::map<std::string, vex::motor &> motors);
00197      void update(bool was_pressed, int x, int y) override;
00198      void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00199
00200  private:
00201      void draw_motor_stats(const std::string &name, vex::motor &mot, unsigned int frame, int x, int
00202 y, vex::brain::lcd &scr);
00203
00204      std::map<std::string, vex::motor &> motors;
00205      static const int y_start = 0;
00206      static const int per_column = 4;
00207      static const int row_height = 20;
00208      static const int row_width = 200;
00209  };
00210
00211  class OdometryPage : public Page
00212  {
00213  public:
00214      OdometryPage(OdometryBase &odom, double robot_width, double robot_height, bool do_trail);
00215      void update(bool was_pressed, int x, int y) override;
00216      void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00217
00218  private:
00219      static const int path_len = 40;
00220      static constexpr char const *field_filename = "vex_field_240p.png";
00221
00222      OdometryBase &odom;
00223      double robot_width;
00224      double robot_height;
00225      uint8_t *buf = nullptr;
00226      int buf_size = 0;
00227      pose_t path[path_len];
00228      int path_index = 0;
00229      bool do_trail;
00230      GraphDrawer velocity_graph;
00231  };
00232
00233  class FunctionPage : public Page
00234  {
00235  public:
00236      FunctionPage(update_func_t update_f, draw_func_t draw_t);
00237      void update(bool was_pressed, int x, int y) override;
00238      void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00239
00240  private:
00241      update_func_t update_f;
00242      draw_func_t draw_f;
00243  };
00244
00245  class PIDPage : public Page
00246  {
00247  public:
00248      PIDPage(
00249          PID &pid, std::string name, std::function<void(void)> onchange = []() {});
00250      PIDPage(
00251          PIDFF &pidff, std::string name, std::function<void(void)> onchange = []() {});
00252
00253      void update(bool was_pressed, int x, int y) override;
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
019
```

```

00290     void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00291
00292     private:
00293     void zero_d_f() { cfg.d = 0; }
00294     void zero_i_f() { cfg.i = 0; }
00295
00296     PID::pid_config_t &cfg;
00297     PID &pid;
00298     const std::string name;
00299     std::function<void(void)> onchange;
00300
00301     SliderWidget p_slider;
00302     SliderWidget i_slider;
00303     SliderWidget d_slider;
00304     ButtonWidget zero_i;
00305     ButtonWidget zero_d;
00306
00307     GraphDrawer graph;
00308
00309 };
00310
00311
00312 }
```

## 6.11 tank\_drive.h

```

00001 #pragma once
00002
00003 #ifndef PI
00004 #define PI 3.141592654
00005 #endif
00006
00007 #include "vex.h"
00008 #include "../core/include/subsystems/odometry/odometry_tank.h"
00009 #include "../core/include/utils/controls/pid.h"
00010 #include "../core/include/utils/controls/feedback_base.h"
00011 #include "../core/include/robot_specs.h"
00012 #include "../core/include/utils/pure_pursuit.h"
00013 #include "../core/include/utils/command_structure/auto_command.h"
00014 #include <vector>
00015
00016 using namespace vex;
00017
00022 class TankDrive
00023 {
00024     public:
00025     enum class BrakeType
00026     {
00027         None,
00028         ZeroVelocity,
00029         Smart,
00030     };
00038     TankDrive(motor_group &left_motors, motor_group &right_motors, robot_specs_t &config, OdometryBase
00039     *odom = NULL);
00040     AutoCommand *DriveToPointCmd(point_t pt, vex::directionType dir = vex::forward, double max_speed =
00041     1.0, double end_speed = 0.0);
00041     AutoCommand *DriveToPointCmd(Feedback &fb, point_t pt, vex::directionType dir = vex::forward, double
00042     max_speed = 1.0, double end_speed = 0.0);
00042
00043     AutoCommand *DriveForwardCmd(double dist, vex::directionType dir = vex::forward, double max_speed =
00044     1.0, double end_speed = 0.0);
00044     AutoCommand *DriveForwardCmd(Feedback &fb, double dist, vex::directionType dir = vex::forward,
00045     double max_speed = 1.0, double end_speed = 0.0);
00046     AutoCommand *TurnToHeadingCmd(double heading, double max_speed = 1.0, double end_speed = 0.0);
00047     AutoCommand *TurnToHeadingCmd(Feedback &fb, double heading, double max_speed = 1.0, double end_speed
00048     = 0.0);
00048
00049     AutoCommand *TurnDegreesCmd(double degrees, double max_speed = 1.0, double start_speed = 0.0);
00050     AutoCommand *TurnDegreesCmd(Feedback &fb, double degrees, double max_speed = 1.0, double end_speed =
00051     0.0);
00052     AutoCommand *PurePursuitCmd(PurePursuit::Path path, directionType dir, double max_speed = 1, double
00053     end_speed = 0);
00053     AutoCommand *PurePursuitCmd(Feedback &feedback, PurePursuit::Path path, directionType dir, double
00054     max_speed = 1, double end_speed = 0);
00055
00058     void stop();
00059
00070     void drive_tank(double left, double right, int power = 1, BrakeType bt = BrakeType::None);
00076     void drive_tank_raw(double left, double right);
00077
00089     void drive_arcade(double forward_back, double left_right, int power = 1, BrakeType bt =
00090     BrakeType::None);
```

```

00090
00102     bool drive_forward(double inches, directionType dir, Feedback &feedback, double max_speed = 1,
00103     double end_speed = 0);
00104
00113     bool drive_forward(double inches, directionType dir, double max_speed = 1, double end_speed = 0);
00114
00125     bool turn_degrees(double degrees, Feedback &feedback, double max_speed = 1, double end_speed = 0);
00126
00137     bool turn_degrees(double degrees, double max_speed = 1, double end_speed = 0);
00138
00151     bool drive_to_point(double x, double y, vex::directionType dir, Feedback &feedback, double max_speed
00152     = 1, double end_speed = 0);
00153
00165     bool drive_to_point(double x, double y, vex::directionType dir, double max_speed = 1, double
00166     end_speed = 0);
00167
00176     bool turn_to_heading(double heading_deg, Feedback &feedback, double max_speed = 1, double end_speed
00177     = 0);
00185     bool turn_to_heading(double heading_deg, double max_speed = 1, double end_speed = 0);
00186
00190     void reset_auto();
00191
00200     static double modify_inputs(double input, int power = 2);
00201
00214     bool pure_pursuit(PurePursuit::Path path, directionType dir, Feedback &feedback, double max_speed =
00215     1, double end_speed = 0);
00216
00229     bool pure_pursuit(PurePursuit::Path path, directionType dir, double max_speed = 1, double end_speed
00230     = 0);
00231 private:
00232     motor_group &left_motors;
00233     motor_group &right_motors;
00234
00235     PID correction_pid;
00236     Feedback *drive_default_feedback = NULL;
00237     Feedback *turn_default_feedback = NULL;
00238
00239     OdometryBase *odometry;
00240
00241     robot_specs_t &config;
00242
00243     bool func_initialized = false;
00244     bool is_pure_pursuit = false;
00245 };

```

## 6.12 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 };

```

## 6.13 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;
00043     virtual bool run() { return true; }
00047     virtual void on_timeout() {}
00048     AutoCommand *withTimeout(double t_seconds)
00049     {
00050         if (this->timeout_seconds < 0)
00051         {
00052             // should never be timed out
00053             return this;
00054         }
00055         this->timeout_seconds = t_seconds;
00056         return this;
00057     }
00058     AutoCommand *withCancelCondition(Condition *true_to_end)
00059     {
00060         this->true_to_end = true_to_end;
00061         return this;
00071     double timeout_seconds = default_timeout;
00072     Condition *true_to_end = nullptr;
00073 };
00074
00079 class FunctionCommand : public AutoCommand
00080 {
00081 public:
00082     FunctionCommand(std::function<bool(void)> f) : f(f) {}
00083     bool run()
00084     {
00085         return f();
00086     }
00087
00088 private:
00089     std::function<bool(void)> f;
00090 };
00091
00092 // Times tested 3
00093 // Test 1 -> false
00094 // Test 2 -> false
00095 // Test 3 -> true
00096 // Returns false until the Nth time that it is called
00097 // This is pretty much only good for implementing RepeatUntil
00098 class TimesTestedCondition : public Condition
00099 {
00100 public:
00101     TimesTestedCondition(size_t N) : max(N) {}
00102     bool test() override
00103     {
00104         count++;
00105         if (count >= max)
00106         {
00107             return true;
00108         }
00109         return false;
00110     }
00111
00112 private:
00113     size_t count = 0;
00114     size_t max;
00115 };
00116
00118 class FunctionCondition : public Condition
00119 {
00120 public:
00121     FunctionCondition(
```

```

00122     std::function<bool()> cond, std::function<void(void)> timeout = []() {} : cond(cond),
00123     {
00124     }
00125     bool test() override;
00126
00127 private:
00128     std::function<bool()> cond;
00129     std::function<void(void)> timeout;
00130 };
00131
00132 class IfTimePassed : public Condition
00133 {
00134 public:
00135     IfTimePassed(double time_s);
00136     bool test() override;
00137
00138 private:
00139     double time_s;
00140     vex::timer tmr;
00141 };
00142
00143
00144 class WaitUntilCondition : public AutoCommand
00145 {
00146 public:
00147     WaitUntilCondition(Condition *cond) : cond(cond) {}
00148     bool run() override
00149     {
00150         return cond->test();
00151     }
00152
00153
00154 private:
00155     Condition *cond;
00156 };
00157
00158
00159 class InOrder : public AutoCommand
00160 {
00161 public:
00162     InOrder(const InOrder &other) = default;
00163     InOrder(std::queue<AutoCommand *> cmdqs);
00164     InOrder(std::initializer_list<AutoCommand *> cmdqs);
00165     bool run() override;
00166     void on_timeout() override;
00167
00168 private:
00169     AutoCommand *current_command = nullptr;
00170     std::queue<AutoCommand *> cmdqs;
00171     vex::timer tmr;
00172 };
00173
00174
00175 class Parallel : public AutoCommand
00176 {
00177 public:
00178     Parallel(std::initializer_list<AutoCommand *> cmdqs);
00179     bool run() override;
00180     void on_timeout() override;
00181
00182 private:
00183     std::vector<AutoCommand *> cmdqs;
00184     std::vector<vex::task *> runners;
00185 };
00186
00187
00188 class Branch : public AutoCommand
00189 {
00190 public:
00191     Branch(Condition *cond, AutoCommand *false_choice, AutoCommand *true_choice);
00192     ~Branch();
00193     bool run() override;
00194     void on_timeout() override;
00195
00196 private:
00197     AutoCommand *false_choice;
00198     AutoCommand *true_choice;
00199     Condition *cond;
00200     bool choice = false;
00201     bool chosen = false;
00202     vex::timer tmr;
00203 };
00204
00205
00206 class Async : public AutoCommand
00207 {
00208 public:
00209     Async(AutoCommand *cmd) : cmd(cmd) {}
00210     bool run() override;
00211
00212 private:
00213
00214
00215
00216
00217
00218
00219
00220
00221

```

```

00222     AutoCommand *cmd = nullptr;
00223 };
00224
00225 class RepeatUntil : public AutoCommand
00226 {
00227 public:
00228     RepeatUntil(InOrder cmds, size_t repeats);
00229     RepeatUntil(InOrder cmds, Condition *true_to_end);
00230     bool run() override;
00231     void on_timeout() override;
00232
00233 private:
00234     const InOrder cmds;
00235     InOrder *working_cmds;
00236     Condition *cond;
00237 };
00238

```

## 6.14 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     //Enumurator 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 };

```

## 6.15 command\_controller.h

00001

```

00010 #pragma once
00011 #include <vector>
00012 #include <queue>
00013 #include "../core/include/utils/command_structure/auto_command.h"
00014
00015 class CommandController
00016 {
00017 public:
00018     [[deprecated("Use list constructor instead.")]] CommandController() : command_queue({}) {}
00019
00020     CommandController(std::initializer_list<AutoCommand *> cmdbs) : command_queue(cmdbs) {}
00021     [[deprecated("Use list constructor instead. If you need to make a decision before adding new
00022 commands, use Branch (https://github.com/RIT-VEX-U/Core/wiki/3-%7C-Utilites#commandcontroller)")]
00023     void add(std::vector<AutoCommand *> cmdbs);
00024     void add(AutoCommand *cmd, double timeout_seconds = 10.0);
00025
00026     [[deprecated("Use list constructor instead. If you need to make a decision before adding new
00027 commands, use Branch (https://github.com/RIT-VEX-U/Core/wiki/3-%7C-Utilites#commandcontroller)")]
00028     void add(std::vector<AutoCommand *> cmdbs, double timeout_sec);
00029     void add_delay(int ms);
00030
00031     void add_cancel_func(std::function<bool(void)> true_if_cancel);
00032
00033     void run();
00034
00035     bool last_command_timed_out();
00036
00037 private:
00038     std::queue<AutoCommand *> command_queue;
00039     bool command_timed_out = false;
00040     std::function<bool()> should_cancel = []() {
00041         return false;
00042     };
00043 };

```

## 6.16 delay\_command.h

```

00001
00008 #pragma once
00009
00010 #include "../core/include/utils/command_structure/auto_command.h"
00011
00012 class DelayCommand: public AutoCommand {
00013     public:
00014         DelayCommand(int ms): ms(ms) {}
00015
00016         bool run() override {
00017             vexDelay(ms);
00018             return true;
00019         }
00020
00021     private:
00022         // amount of milliseconds to wait
00023         int ms;
00024     };

```

## 6.17 drive\_commands.h

```

00001
00019 #pragma once
00020
00021 #include "vex.h"
00022 #include "../core/include/utils/geometry.h"
00023 #include "../core/include/utils/command_structure/auto_command.h"
00024 #include "../core/include/subsystems/tank_drive.h"
00025
00026 using namespace vex;
00027
00028
00029 // ===== DRIVING =====
00030
00031
00036 class DriveForwardCommand: public AutoCommand
00037 {
00038     public:
00039         DriveForwardCommand(TankDrive &drive_sys, Feedback &feedback, double inches, directionType dir,
00040         double max_speed=1, double end_speed=0);
00041
00042         bool run() override;
00043         void on_timeout() override;

```

```
00051
00052     private:
00053         // drive system to run the function on
00054         TankDrive &drive_sys;
00055
00056         // feedback controller to use
00057         Feedback &feedback;
00058
00059         // parameters for drive_forward
00060         double inches;
00061         directionType dir;
00062         double max_speed;
00063         double end_speed;
00064     };
00065
00070     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);
00075
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:
00180     PurePursuitCommand(TankDrive &drive_sys, Feedback &feedback, PurePursuit::Path path, directionType
00181     dir, double max_speed=1, double end_speed=0);
00182
00183     bool run() override;
00184
00185     void on_timeout() override;
00186
00187     private:
00188     TankDrive &drive_sys;
00189     PurePursuit::Path path;
00190     directionType dir;
00191     Feedback &feedback;
00192     double max_speed;
00193     double end_speed;
00194
00195 };
00196
00197
00198
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208 };
00209
00210
00211
00212
00213
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
00228
00229
00230
00231
00232
00233 // ===== ODOMETRY =====
00234
00235
00236
00237
00238
00239 class OdomSetPosition: public AutoCommand
00240 {
00241     public:
00242     OdomSetPosition(OdometryBase &odom, const pose_t &newpos=OdometryBase::zero_pos);
00243
00244     bool run() override;
00245
00246     private:
00247     // drive system with an odometry config
00248     OdometryBase &odom;
00249     pose_t newpos;
00250 };
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260 };

```

## 6.18 flywheel\_commands.h

```

00001
00002 #pragma once
00003
00004
00005 #include "../core/include/subsystems/flywheel.h"
00006 #include "../core/include/utils/command_structure/auto_command.h"
00007
00008
00009
00010
00011
00012
00013
00014
00015 class SpinRPMCommand: public AutoCommand {
00016     public:
00017     SpinRPMCommand(Flywheel &flywheel, int rpm);
00018
00019     bool run() override;
00020
00021     private:
00022     // Flywheel instance to run the function on
00023     Flywheel &flywheel;
00024
00025     // parameters for spin_rpm
00026     int rpm;
00027 };
00028
00029
00030
00031
00032
00033
00034
00035
00036
00037
00038
00039
00040
00041
00042
00043
00044
00045 class WaitUntilUpToSpeedCommand: public AutoCommand {
00046     public:
00047     WaitUntilUpToSpeedCommand(Flywheel &flywheel, int threshold_rpm);
00048
00049     bool run() override;
00050
00051     private:
00052     // Flywheel instance to run the function on
00053     Flywheel &flywheel;
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063

```

```

00064     // if the actual speed is equal to the desired speed +/- this value, we are ready to fire
00065     int threshold_rpm;
00066 };
00068
00074 class FlywheelStopCommand: public AutoCommand {
00075 public:
00076     FlywheelStopCommand(Flywheel &flywheel);
00077
00078     bool run() override;
00079
00089 private:
00090     // Flywheel instance to run the function on
00091     Flywheel &flywheel;
00092 };
00093
00099 class FlywheelStopMotorsCommand: public AutoCommand {
00100 public:
00101     FlywheelStopMotorsCommand(Flywheel &flywheel);
00102
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 };

```

## 6.19 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;
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 };

```

## 6.20 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 };

```

## 6.21 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
00041     typedef struct
00042     {
00043         double kS;
00044         double KV;
00045         double kA;
00046         double KG;
00047     } ff_config_t;
00048
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;
00080
00081 };
00082
00083
00091 FeedForward::ff_config_t tune_feedforward(vex::motor_group &motor, double pct, double duration);

```

## 6.22 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/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     double update(double sensor_val) override;
00061
00062     double get() override;
00063
00064     void set_limits(double lower, double upper) override;
00065
00066     bool is_on_target() override;
00067
00068     motion_t get_motion() const;
00069
00070     screen::Page *Page();
00071
00072     static FeedForward::ff_config_t tune_feedforward(TankDrive &drive, OdometryTank &odometry, double
00073     pct=0.6, double duration=2);
00074
00075     private:
00076
00077     m_profile_cfg_t config;
00078
00079     PID pid;
00080     FeedForward ff;
00081     TrapezoidProfile profile;
00082
00083     double current_pos;
00084     double end_pt;
00085
00086     double lower_limit = 0, upper_limit = 0;
00087     double out = 0;
00088     motion_t cur_motion;
00089
00090     vex::timer tmr;
00091     friend class MotionControllerPage;
00092
00093 };
00094

```

## 6.23 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

```

```

00135 void set_target(double target);
00136
00137 pid_config_t
00138 &config;
00140
00141 private:
00142 double last_error =
00143 0;
00144 double accum_error =
00145 0;
00146
00147 double last_time = 0;
00148 double on_target_last_time =
00149 0;
00150
00151 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 };

```

## 6.24 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
0018 void init(double start_pt, double set_pt, double start_vel,
0019           double end_vel) override;
0020
0025 void set_target(double set_pt);
0026
0027 double get_target() const;
0028 double get_sensor_val() const;
0036 double update(double val) override;
0037
0046 double update(double val, double vel_setpt, double a_setpt = 0);
0047
0051 double get() override;
0052
0060 void set_limits(double lower, double upper) override;
0061
0065 bool is_on_target() override;
0066
0067 void reset();
0068
0069 PID pid;
0070
0071 private:
0072 FeedForward::ff_config_t &ff_cfg;
0073
0074 FeedForward ff;
0075
0076 double out;
0077 double lower_lim, upper_lim;
0078 };

```

## 6.25 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);
0011     void init(double start_pt, double set_pt, double, double);
0012     double update(double val) override;
0013
0014     double get() override;
0015
0016     void set_limits(double lower, double upper) override;
0017
0018     bool is_on_target() override;
0019
0020     double TBH_gain;
0021     double first_cross_split;
0022
0023 private:
0024     double on_target_threshold;
0025
0026     double target = 0.0;
0027
0028     bool first_cross = true;
0029     double tbh = 0.0;
0030     double prev_error = 0.0;
0031
0032     double output = 0.0;
0033     double lower = 0.0, upper = 0.0;
0034 };

```

## 6.26 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,

```

```

00164                               double v_target);
00165
00173     trapezoid_profile_segment_t calculate_next_segment(double s, double v);
00174 };

```

## 6.27 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
00014 class GenericAuto
00015 {
00016     public:
00017
00031     [[deprecated("Use CommandController instead.")]]
00032     bool run(bool blocking);
00033
00038     [[deprecated("Use CommandController instead.")]]
00039     void add(state_ptr new_state);
00040
00045     [[deprecated("Use CommandController instead.")]]
00046     void add_async(state_ptr async_state);
00047
00052     [[deprecated("Use CommandController instead.")]]
00053     void add_delay(int ms);
00054
00055     private:
00056
00057     std::queue<state_ptr> state_list;
00058
00059 };

```

## 6.28 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) + std::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     }
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 };

```

## 6.29 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 };

```

## 6.30 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
00020 class Logger
00021 {
00022 private:
00023     const std::string filename;
00024     vex::brain::sdcard sd;
00025     void write_level(LogLevel l);
00026
00027 public:
00029     static constexpr int MAX_FORMAT_LEN = 512;
00032     explicit Logger(const std::string &filename);
00033
00035     Logger(const Logger &l) = delete;
00037     Logger &operator=(const Logger &l) = delete;
00038
00042     void Log(const std::string &s);
00043
00047     void Log(LogLevel level, const std::string &s);
00048
00051     void Logln(const std::string &s);
00052
00056     void Logln(LogLevel level, const std::string &s);
00057
00061     void Logf(const char *fmt, ...);
00062
00067     void Logf(LogLevel level, const char *fmt, ...);
00068 };

```

## 6.31 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);

```

## 6.32 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;
00056
00057 int get_size() const;
00058
00059 private:
00060     int buffer_index;           // index of the next value to be overridden
00061     std::vector<double> buffer; // all current data readings we've taken
00062     double current_avg;        // the current value of the data
00063 };
00064
00065 class ExponentialMovingAverage : public Filter
00066 {
00067     public:
00068     /*
00069     * Create a moving average calculator with 0 as the default value
00070     *
00071     * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
00072     reading
00073     */
00074     ExponentialMovingAverage(int buffer_size);
00075     /*
00076     * Create a moving average calculator with a specified default value
00077     * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
00078     reading
00079     * @param starting_value The value that the average will be before any data is added
00080     */
00081     ExponentialMovingAverage(int buffer_size, double starting_value);
00082
00083     /*
00084     * Add a reading to the buffer
00085     * Before:
00086     * [ 1 1 2 2 3 3 ] => 2
00087     * ^
00088     * After:
00089     * [ 2 1 2 2 3 3 ] => 2.16
00090     * ^
00091     * @param n  the sample that will be added to the moving average.
00092     */
00093     void add_entry(double n) override;
00094
00095     double get_value() const override;
00096
00097     int get_size();
00098
00099     private:
00100     int buffer_index;           // index of the next value to be overridden
00101     std::vector<double> buffer; // all current data readings we've taken
00102     double current_avg;        // the current value of the data
00103 };

```

## 6.33 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
00016             std::vector<point_t> get_points();
00017
00018             double get_radius();
00019
00020             bool is_valid();
00021
00022         private:
00023             std::vector<point_t> points;
00024             double radius;
00025             bool valid;
00026     };
00027 }
```

```

00048     struct spline
00049     {
00050         double a, b, c, d, x_start, x_end;
00051
00052         double getY(double x) {
00053             return a * pow((x - x_start), 3) + b * pow((x - x_start), 2) + c * (x - x_start) + d;
00054         }
00055     };
00060     struct hermite_point
00061     {
00062         double x;
00063         double y;
00064         double dir;
00065         double mag;
00066
00067         point_t getPoint() const {
00068             return {x, y};
00069         }
00070
00071         Vector2D getTangent() const {
00072             return Vector2D(dir, mag);
00073         }
00074     };
00075
00080     extern std::vector<point_t> line_circle_intersections(point_t center, double r, point_t point1,
00081     point_t point2);
00084     extern point_t get_loookahead(const std::vector<point_t> &path, pose_t robot_loc, double radius);
00085
00089     extern std::vector<point_t> inject_path(const std::vector<point_t> &path, double spacing);
00090
00102     extern std::vector<point_t> smooth_path(const std::vector<point_t> &path, double weight_data, double
00103     weight_smooth, double tolerance);
00104
00105     extern std::vector<point_t> smooth_path_cubic(const std::vector<point_t> &path, double res);
00106
00114     extern std::vector<point_t> smooth_path_hermite(const std::vector<hermite_point> &path, double
00115     step);
00116
00126     extern double estimate_remaining_dist(const std::vector<point_t> &path, pose_t robot_pose, double
00127     radius);
00127
00128 }
```

## 6.34 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
00010 const char serialization_separator = '$';
00012 const std::size_t MAX_FILE_SIZE = 4096;
00013
00015 class Serializer
00016 {
00017     private:
00018         bool flush_always;
00019         std::string filename;
00020         std::map<std::string, int> ints;
00021         std::map<std::string, bool> bools;
00022         std::map<std::string, double> doubles;
00023         std::map<std::string, std::string> strings;
00024
00026     bool read_from_disk();
00027
00028 public:
00030     ~Serializer()
00031     {
00032         save_to_disk();
00033         printf("Saving %s\n", filename.c_str());
00034         fflush(stdout);
00035     }
00036
00040     explicit Serializer(const std::string &filename, bool flush_always = true) :
00041         flush_always(flush_always), filename(filename), ints({}), bools({}), doubles({}), strings({})
00042     {
00043         read_from_disk();
00044     }
00045 }
```

```

00047     void save_to_disk() const;
00048
00050
00054     void set_int(const std::string &name, int i);
00055
00059     void set_bool(const std::string &name, bool b);
00060
00064     void set_double(const std::string &name, double d);
00065
00069     void set_string(const std::string &name, std::string str);
00070
00073
00078     int int_or(const std::string &name, int otherwise);
00079
00084     bool bool_or(const std::string &name, bool otherwise);
00085
00090     double double_or(const std::string &name, double otherwise);
00091
00096     std::string string_or(const std::string &name, std::string otherwise);
00097 };

```

## 6.35 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
00015 class Vector2D
00016 {
00017 public:
00024     Vector2D(double dir, double mag);
00025
00031     Vector2D(point_t p);
00032
00040     double get_dir() const;
00041
00045     double get_mag() const;
00046
00050     double get_x() const;
00051
00055     double get_y() const;
00056
00061     Vector2D normalize();
00062
00067     point_t point();
00068
00074     Vector2D operator*(const double &x);
00081     Vector2D operator+(const Vector2D &other);
00088     Vector2D operator-(const Vector2D &other);
00089
00090 private:
00091     double dir, mag;
00093
00094 };
00095
00101 double deg2rad(double deg);
00102
00109 double rad2deg(double r);

```

# Index

accel  
    OdometryBase, 118

add  
    CommandController, 36, 37  
    GenericAuto, 78

add\_async  
    GenericAuto, 79

add\_cancel\_func  
    CommandController, 38

add\_delay  
    CommandController, 38  
    GenericAuto, 79

add\_entry  
    ExponentialMovingAverage, 56  
    Filter, 62  
    MovingAverage, 107

add\_samples  
    GraphDrawer, 80, 81

AndCondition, 13  
    test, 14

ang\_accel\_deg  
    OdometryBase, 118

ang\_speed\_deg  
    OdometryBase, 118

Async, 14  
    run, 16

auto\_drive  
    MecanumDrive, 96

auto\_turn  
    MecanumDrive, 96

AutoChooser, 16  
    AutoChooser, 17  
    choice, 19  
    draw, 18  
    get\_choice, 18  
    list, 19  
    update, 18

AutoChooser::entry\_t, 53  
    name, 54

AutoCommand, 20  
    on\_timeout, 21  
    run, 21  
    timeout\_seconds, 22

background\_task  
    OdometryBase, 115

BangBang, 22  
    get, 23  
    init, 23  
    is\_on\_target, 24

set\_limits, 24  
    update, 24

BasicSolenoidSet, 25  
    BasicSolenoidSet, 26  
    run, 26

BasicSpinCommand, 27  
    BasicSpinCommand, 28  
    run, 29

BasicStopCommand, 29  
    BasicStopCommand, 31  
    run, 31

bool\_or  
    Serializer, 157

BrakeType  
    TankDrive, 173

Branch, 32  
    on\_timeout, 33  
    run, 33

ButtonWidget  
    screen::ButtonWidget, 34, 35

calculate  
    FeedForward, 60  
    TrapezoidProfile, 186

calculate\_time\_based  
    TrapezoidProfile, 188

choice  
    AutoChooser, 19

CommandController, 36  
    add, 36, 37  
    add\_cancel\_func, 38  
    add\_delay, 38  
    CommandController, 36  
    last\_command\_timed\_out, 38  
    run, 38

Condition, 39

config  
    PID, 140

control\_continuous  
    Lift< T >, 87

control\_manual  
    Lift< T >, 87

control\_setpoints  
    Lift< T >, 87

Core, 1

current\_pos  
    OdometryBase, 118

CustomEncoder, 39  
    CustomEncoder, 40  
    position, 41

rotation, 41  
 setPosition, 41  
 setRotation, 41  
 velocity, 42  
  
 DelayCommand, 42  
     DelayCommand, 44  
     run, 45  
 dist  
     point\_t, 147  
 double\_or  
     Serializer, 157  
 draw  
     AutoChooser, 18  
     FlywheelPage, 67  
     GraphDrawer, 81  
     MotionControllerPage, 105  
     screen::FunctionPage, 77  
     screen::OdometryPage, 121  
     screen::Page, 130  
     screen::PIDPage, 146  
     screen::StatsPage, 167  
     screen::WidgetPage, 204  
 drive  
     MecanumDrive, 97  
 drive\_ arcade  
     TankDrive, 174  
 drive\_ forward  
     TankDrive, 174, 175  
 drive\_ raw  
     MecanumDrive, 97  
 drive\_ tank  
     TankDrive, 176  
 drive\_ tank\_ raw  
     TankDrive, 176  
 drive\_ to\_ point  
     TankDrive, 176, 177  
 DriveForwardCommand, 45  
     DriveForwardCommand, 47  
     on\_timeout, 47  
     run, 47  
 DriveStopCommand, 48  
     DriveStopCommand, 49  
     on\_timeout, 49  
     run, 50  
 DriveToPointCommand, 50  
     DriveToPointCommand, 52  
     run, 52  
  
 end\_async  
     OdometryBase, 115  
 error\_method  
     PID::pid\_config\_t, 140  
 ERROR\_TYPE  
     PID, 136  
 ExponentialMovingAverage, 54  
     add\_entry, 56  
     ExponentialMovingAverage, 55  
     get\_size, 56  
  
     get\_value, 56  
  
 Feedback, 57  
     get, 57  
     init, 57  
     is\_on\_target, 58  
     set\_limits, 58  
     update, 58  
 FeedForward, 59  
     calculate, 60  
     FeedForward, 59  
 FeedForward::ff\_config\_t, 60  
     kA, 61  
     kG, 61  
     kS, 61  
     kV, 61  
 Filter, 61  
     add\_entry, 62  
     get\_value, 62  
 Flywheel, 62  
     Flywheel, 63  
     get\_motors, 63  
     get\_target, 63  
     getRPM, 64  
     is\_on\_target, 64  
     Page, 64  
     spin\_manual, 64  
     spin\_rpm, 65  
     SpinRpmCmd, 65  
     spinRPMTask, 66  
     stop, 65  
     WaitUntilUpToSpeedCmd, 66  
 FlywheelPage, 66  
     draw, 67  
     update, 67  
 FlywheelStopCommand, 68  
     FlywheelStopCommand, 69  
     run, 69  
 FlywheelStopMotorsCommand, 70  
     FlywheelStopMotorsCommand, 71  
     run, 71  
 FlywheelStopNonTasksCommand, 72  
 FunctionCommand, 73  
     run, 74  
 FunctionCondition, 75  
     test, 76  
 FunctionPage  
     screen::FunctionPage, 77  
  
     GenericAuto, 78  
         add, 78  
         add\_async, 79  
         add\_delay, 79  
         run, 79  
  
     get  
         BangBang, 23  
         Feedback, 57  
         MotionController, 101  
         PID, 137

PIDFF, 142  
TakeBackHalf, 169  
get\_accel  
    OdometryBase, 115  
get-angular\_accel\_deg  
    OdometryBase, 115  
get-angular\_speed\_deg  
    OdometryBase, 115  
get\_async  
    Lift< T >, 88  
get\_choice  
    AutoChooser, 18  
get\_dir  
    Vector2D, 196  
get\_error  
    PID, 137  
get\_mag  
    Vector2D, 196  
get\_motion  
    MotionController, 101  
get\_motors  
    Flywheel, 63  
get\_movement\_time  
    TrapezoidProfile, 188  
get\_points  
    PurePursuit::Path, 134  
get\_position  
    OdometryBase, 116  
get\_radius  
    PurePursuit::Path, 135  
get\_sensor\_val  
    PID, 137  
get\_setpoint  
    Lift< T >, 88  
get\_size  
    ExponentialMovingAverage, 56  
    MovingAverage, 107  
get\_speed  
    OdometryBase, 116  
get\_target  
    Flywheel, 63  
    PID, 137  
get\_value  
    ExponentialMovingAverage, 56  
    Filter, 62  
    MovingAverage, 108  
get\_x  
    Vector2D, 196  
get\_y  
    Vector2D, 197  
getRPM  
    Flywheel, 64  
GraphDrawer, 80  
    add\_samples, 80, 81  
    draw, 81  
    GraphDrawer, 80  
handle  
    OdometryBase, 119  
hold  
    Lift< T >, 88  
home  
    Lift< T >, 88  
IfTimePassed, 82  
    test, 83  
include/robot\_specs.h, 207  
include/subsystems/custom\_encoder.h, 207  
include/subsystems/flywheel.h, 208  
include/subsystems/layout.h, 208  
include/subsystems/lift.h, 209  
include/subsystems/mecanum\_drive.h, 211  
include/subsystems/odometry/odometry\_3wheel.h, 212  
include/subsystems/odometry/odometry\_base.h, 213  
include/subsystems/odometry/odometry\_tank.h, 213  
include/subsystems/screen.h, 214  
include/subsystems/tank\_drive.h, 217  
include/utils/auto\_chooser.h, 218  
include/utils/command\_structure/auto\_command.h, 219  
include/utils/command\_structure/basic\_command.h,  
    221  
include/utils/command\_structure/command\_controller.h,  
    221  
include/utils/command\_structure/delay\_command.h,  
    222  
include/utils/command\_structure/drive\_commands.h,  
    222  
include/utils/command\_structure/flywheel\_commands.h,  
    224  
include/utils/controls/bang\_bang.h, 225  
include/utils/controls/feedback\_base.h, 225  
include/utils/controls/feedforward.h, 226  
include/utils/controls/motion\_controller.h, 226  
include/utils/controls/pid.h, 227  
include/utils/controls/pidff.h, 228  
include/utils/controls/take\_back\_half.h, 228  
include/utils/controls/trapezoid\_profile.h, 229  
include/utils/generic\_auto.h, 230  
include/utils/geometry.h, 230  
include/utils/graph\_drawer.h, 231  
include/utils/logger.h, 232  
include/utils/math\_util.h, 232  
include/utils/moving\_average.h, 233  
include/utils/pure\_pursuit.h, 234  
include/utils/serializer.h, 235  
include/utils/vector2d.h, 236  
init  
    BangBang, 23  
    Feedback, 57  
    MotionController, 102  
    PID, 138  
    PIDFF, 142  
    TakeBackHalf, 169  
InOrder, 83  
    on\_timeout, 85  
    run, 85  
int\_or  
    Serializer, 158

is\_on\_target  
 BangBang, 24  
 Feedback, 58  
 Flywheel, 64  
 MotionController, 102  
 PID, 138  
 PIDFF, 143  
 TakeBackHalf, 170  
 is\_valid  
 PurePursuit::Path, 135

kA  
 FeedForward::ff\_config\_t, 61

kG  
 FeedForward::ff\_config\_t, 61

kS  
 FeedForward::ff\_config\_t, 61

kV  
 FeedForward::ff\_config\_t, 61

last\_command\_timed\_out  
 CommandController, 38

Lift  
 Lift< T >, 86  
 Lift< T >, 85  
 control\_continuous, 87  
 control\_manual, 87  
 control\_setpoints, 87  
 get\_async, 88  
 get\_setpoint, 88  
 hold, 88  
 home, 88  
 Lift, 86  
 set\_async, 88  
 set\_position, 88  
 set\_sensor\_function, 89  
 set\_sensor\_reset, 89  
 set\_setpoint, 89  
 Lift< T >::lift\_cfg\_t, 90

list  
 AutoChooser, 19

Log  
 Logger, 92

Logf  
 Logger, 92

Logger, 91  
 Log, 92  
 Logf, 92  
 Logger, 91  
 LogIn, 93

LogIn  
 Logger, 93

Mat2, 94

MecanumDrive, 95  
 auto\_drive, 96  
 auto\_turn, 96  
 drive, 97  
 drive\_raw, 97

MecanumDrive, 95  
 MecanumDrive::mecanumdrive\_config\_t, 98  
 modify\_inputs  
 TankDrive, 178  
 motion\_t, 99  
 MotionController, 99  
 get, 101  
 get\_motion, 101  
 init, 102  
 is\_on\_target, 102  
 MotionController, 101  
 set\_limits, 102  
 tune\_feedforward, 103  
 update, 103

MotionController::m\_profile\_cfg\_t, 93

MotionControllerPage, 104  
 draw, 105  
 update, 105

MovingAverage, 106  
 add\_entry, 107  
 get\_size, 107  
 get\_value, 108  
 MovingAverage, 107

mut  
 OdometryBase, 119

name  
 AutoChooser::entry\_t, 54

None  
 TankDrive, 173

normalize  
 Vector2D, 197

Odometry3Wheel, 108  
 Odometry3Wheel, 110  
 tune, 111  
 update, 111

Odometry3Wheel::odometry3wheel\_cfg\_t, 112  
 off\_axis\_center\_dist, 112  
 wheel\_diam, 112  
 wheelbase\_dist, 112

OdometryBase, 113  
 accel, 118  
 ang\_accel\_deg, 118  
 ang\_speed\_deg, 118  
 background\_task, 115  
 current\_pos, 118  
 end\_async, 115  
 get\_accel, 115  
 get\_angular\_accel\_deg, 115  
 get\_angular\_speed\_deg, 115  
 get\_position, 116  
 get\_speed, 116  
 handle, 119  
 mut, 119

OdometryBase, 114  
 pos\_diff, 116  
 rot\_diff, 117  
 set\_position, 117

smallest\_angle, 117  
speed, 119  
update, 118  
zero\_pos, 119  
OdometryPage  
  screen::OdometryPage, 121  
OdometryTank, 122  
  OdometryTank, 123, 124  
  set\_position, 125  
  update, 125  
OdomSetPosition, 125  
  OdomSetPosition, 127  
  run, 128  
off\_axis\_center\_dist  
  Odometry3Wheel::odometry3wheel\_cfg\_t, 112  
on\_target\_time  
  PID::pid\_config\_t, 140  
on\_timeout  
  AutoCommand, 21  
  Branch, 33  
  DriveForwardCommand, 47  
  DriveStopCommand, 49  
  InOrder, 85  
  Parallel, 133  
  PurePursuitCommand, 151  
  RepeatUntil, 154  
  TurnDegreesCommand, 191  
  TurnToHeadingCommand, 194  
operator+  
  point\_t, 147  
  Vector2D, 197  
operator-  
  point\_t, 148  
  Vector2D, 198  
operator\*  
  Vector2D, 197  
OrCondition, 128  
  test, 129  
  
Page  
  Flywheel, 64  
Parallel, 131  
  on\_timeout, 133  
  run, 133  
parallel\_runner\_info, 133  
Path  
  PurePursuit::Path, 134  
PID, 135  
  config, 140  
  ERROR\_TYPE, 136  
  get, 137  
  get\_error, 137  
  get\_sensor\_val, 137  
  get\_target, 137  
  init, 138  
  is\_on\_target, 138  
  PID, 136  
  reset, 138  
  set\_limits, 138  
  set\_target, 139  
  update, 139  
PID::pid\_config\_t, 140  
  error\_method, 140  
  on\_target\_time, 140  
PIDFF, 141  
  get, 142  
  init, 142  
  is\_on\_target, 143  
  set\_limits, 143  
  set\_target, 143  
  update, 143, 144  
PIDPage  
  screen::PIDPage, 145  
point  
  Vector2D, 198  
point\_t, 146  
  dist, 147  
  operator+, 147  
  operator-, 148  
pos\_diff  
  OdometryBase, 116  
pose\_t, 148  
position  
  CustomEncoder, 41  
pure\_pursuit  
  TankDrive, 178, 179  
PurePursuit::hermite\_point, 81  
PurePursuit::Path, 134  
  get\_points, 134  
  get\_radius, 135  
  is\_valid, 135  
  Path, 134  
PurePursuit::spline, 166  
PurePursuitCommand, 149  
  on\_timeout, 151  
  PurePursuitCommand, 150  
  run, 151  
  
Rect, 151  
RepeatUntil, 152  
  on\_timeout, 154  
  RepeatUntil, 153  
  run, 154  
reset  
  PID, 138  
reset\_auto  
  TankDrive, 180  
robot\_specs\_t, 154  
rot\_diff  
  OdometryBase, 117  
rotation  
  CustomEncoder, 41  
run  
  Async, 16  
  AutoCommand, 21  
  BasicSolenoidSet, 26  
  BasicSpinCommand, 29  
  BasicStopCommand, 31

Branch, 33  
 CommandController, 38  
 DelayCommand, 45  
 DriveForwardCommand, 47  
 DriveStopCommand, 50  
 DriveToPointCommand, 52  
 FlywheelStopCommand, 69  
 FlywheelStopMotorsCommand, 71  
 FunctionCommand, 74  
 GenericAuto, 79  
 InOrder, 85  
 OdomSetPosition, 128  
 Parallel, 133  
 PurePursuitCommand, 151  
 RepeatUntil, 154  
 SpinRPMCommand, 165  
 TurnDegreesCommand, 191  
 TurnToHeadingCommand, 194  
 WaitUntilCondition, 200  
 WaitUntilUpToSpeedCommand, 202

save\_to\_disk  
 Serializer, 158

screen::ButtonConfig, 34  
 screen::ButtonWidget, 34  
 ButtonWidget, 34, 35  
 update, 35

screen::CheckboxConfig, 35  
 screen::FunctionPage, 76  
 draw, 77  
 FunctionPage, 77  
 update, 77

screen::LabelConfig, 85  
 screen::OdometryPage, 120  
 draw, 121  
 OdometryPage, 121  
 update, 121

screen::Page, 129  
 draw, 130  
 update, 131

screen::PIDPage, 144  
 draw, 146  
 PIDPage, 145  
 update, 146

screen::ScreenData, 155  
 screen::ScreenRect, 156  
 screen::SizedWidget, 160

screen::SliderConfig, 161  
 screen::SliderWidget, 161  
 SliderWidget, 162  
 update, 163

screen::StatsPage, 166  
 draw, 167  
 StatsPage, 167  
 update, 168

screen::TextConfig, 183  
 screen::WidgetConfig, 203  
 screen::WidgetPage, 204  
 draw, 204

update, 205  
 Serializer, 156  
 bool\_or, 157  
 double\_or, 157  
 int\_or, 158  
 save\_to\_disk, 158  
 Serializer, 157  
 set\_bool, 158  
 set\_double, 159  
 set\_int, 159  
 set\_string, 159  
 string\_or, 159

set\_accel  
 TrapezoidProfile, 188

set\_async  
 Lift< T >, 88

set\_bool  
 Serializer, 158

set\_double  
 Serializer, 159

set\_endpts  
 TrapezoidProfile, 189

set\_int  
 Serializer, 159

set\_limits  
 BangBang, 24  
 Feedback, 58  
 MotionController, 102  
 PID, 138  
 PIDFF, 143  
 TakeBackHalf, 170

set\_max\_v  
 TrapezoidProfile, 189

set\_position  
 Lift< T >, 88  
 OdometryBase, 117  
 OdometryTank, 125

set\_sensor\_function  
 Lift< T >, 89

set\_sensor\_reset  
 Lift< T >, 89

set\_setpoint  
 Lift< T >, 89

set\_string  
 Serializer, 159

set\_target  
 PID, 139  
 PIDFF, 143

set\_vel\_endpts  
 TrapezoidProfile, 189

setPosition  
 CustomEncoder, 41

setRotation  
 CustomEncoder, 41

SliderCfg, 161  
 SliderWidget  
 screen::SliderWidget, 162

smallest\_angle

OdometryBase, 117  
Smart  
    TankDrive, 173  
speed  
    OdometryBase, 119  
spin\_manual  
    Flywheel, 64  
spin\_rpm  
    Flywheel, 65  
SpinRpmCmd  
    Flywheel, 65  
SpinRPMCommand, 163  
    run, 165  
    SpinRPMCommand, 165  
spinRPMTask  
    Flywheel, 66  
StatsPage  
    screen::StatsPage, 167  
stop  
    Flywheel, 65  
    TankDrive, 180  
string\_or  
    Serializer, 159  
TakeBackHalf, 168  
    get, 169  
    init, 169  
    is\_on\_target, 170  
    set\_limits, 170  
    update, 170  
TankDrive, 172  
    BrakeType, 173  
    drive\_arcade, 174  
    drive\_forward, 174, 175  
    drive\_tank, 176  
    drive\_tank\_raw, 176  
    drive\_to\_point, 176, 177  
    modify\_inputs, 178  
    None, 173  
    pure\_pursuit, 178, 179  
    reset\_auto, 180  
    Smart, 173  
    stop, 180  
    TankDrive, 173  
    turn\_degrees, 180, 181  
    turn\_to\_heading, 181, 182  
    ZeroVelocity, 173  
test  
    AndCondition, 14  
    FunctionCondition, 76  
    IfTimePassed, 83  
    OrCondition, 129  
    TimesTestedCondition, 184  
timeout\_seconds  
    AutoCommand, 22  
TimesTestedCondition, 183  
    test, 184  
trapezoid\_profile\_segment\_t, 184  
TrapezoidProfile, 185  
calculate, 186  
calculate\_time\_based, 188  
get\_movement\_time, 188  
set\_accel, 188  
set\_endpts, 189  
set\_max\_v, 189  
set\_vel\_endpts, 189  
TrapezoidProfile, 186  
tune  
    Odometry3Wheel, 111  
tune\_feedforward  
    MotionController, 103  
turn\_degrees  
    TankDrive, 180, 181  
turn\_to\_heading  
    TankDrive, 181, 182  
TurnDegreesCommand, 189  
    on\_timeout, 191  
    run, 191  
    TurnDegreesCommand, 191  
TurnToHeadingCommand, 192  
    on\_timeout, 194  
    run, 194  
    TurnToHeadingCommand, 193  
update  
    AutoChooser, 18  
    BangBang, 24  
    Feedback, 58  
    FlywheelPage, 67  
    MotionController, 103  
    MotionControllerPage, 105  
    Odometry3Wheel, 111  
    OdometryBase, 118  
    OdometryTank, 125  
    PID, 139  
    PIDFF, 143, 144  
    screen::ButtonWidget, 35  
    screen::FunctionPage, 77  
    screen::OdometryPage, 121  
    screen::Page, 131  
    screen::PIDPage, 146  
    screen::SliderWidget, 163  
    screen::StatsPage, 168  
    screen::WidgetPage, 205  
    TakeBackHalf, 170  
Vector2D, 194  
    get\_dir, 196  
    get\_mag, 196  
    get\_x, 196  
    get\_y, 197  
    normalize, 197  
    operator+, 197  
    operator-, 198  
    operator\*, 197  
    point, 198  
    Vector2D, 195, 196  
velocity

CustomEncoder, [42](#)  
WaitUntilCondition, [199](#)  
    run, [200](#)  
WaitUntilUpToSpeedCmd  
    Flywheel, [66](#)  
WaitUntilUpToSpeedCommand, [201](#)  
    run, [202](#)  
    WaitUntilUpToSpeedCommand, [202](#)  
wheel\_diam  
    Odometry3Wheel::odometry3wheel\_cfg\_t, [112](#)  
wheelbase\_dist  
    Odometry3Wheel::odometry3wheel\_cfg\_t, [112](#)  
zero\_pos  
    OdometryBase, [119](#)  
ZeroVelocity  
    TankDrive, [173](#)