

# 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

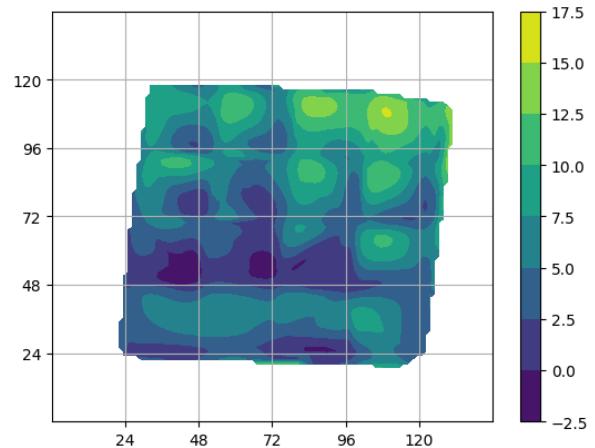


Figure 8 - Error Heat Map

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);

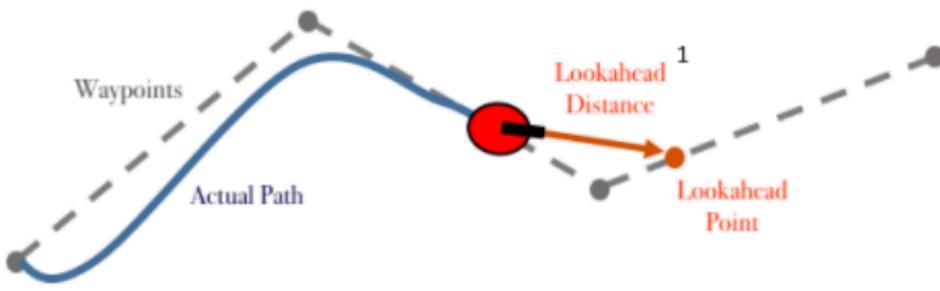
drive_tank(lside, rside);
```

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.

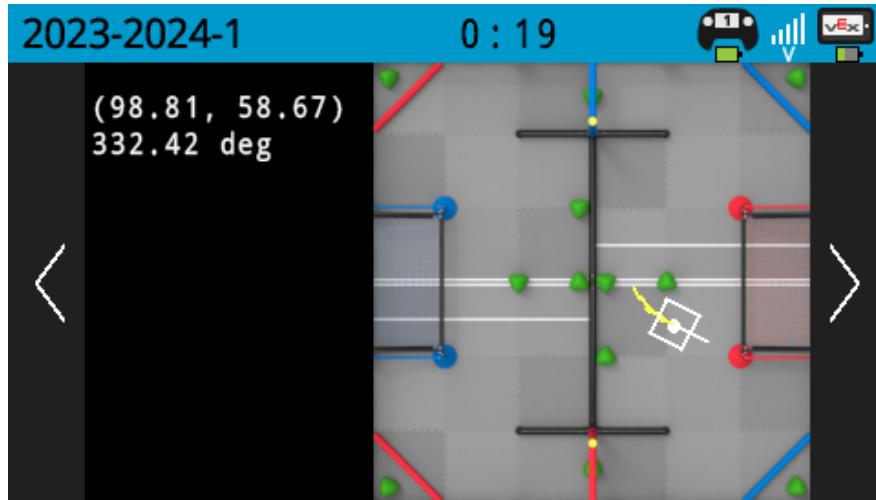


Figure 15: A field display for the Over Under season

### 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.

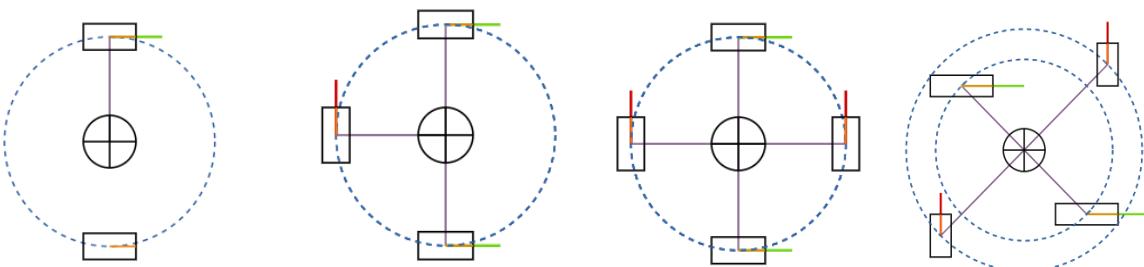


Figure 19: 2 pod, 3 pod, and arbitrary pods such a system could handle.

### 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 ( $r$ ).

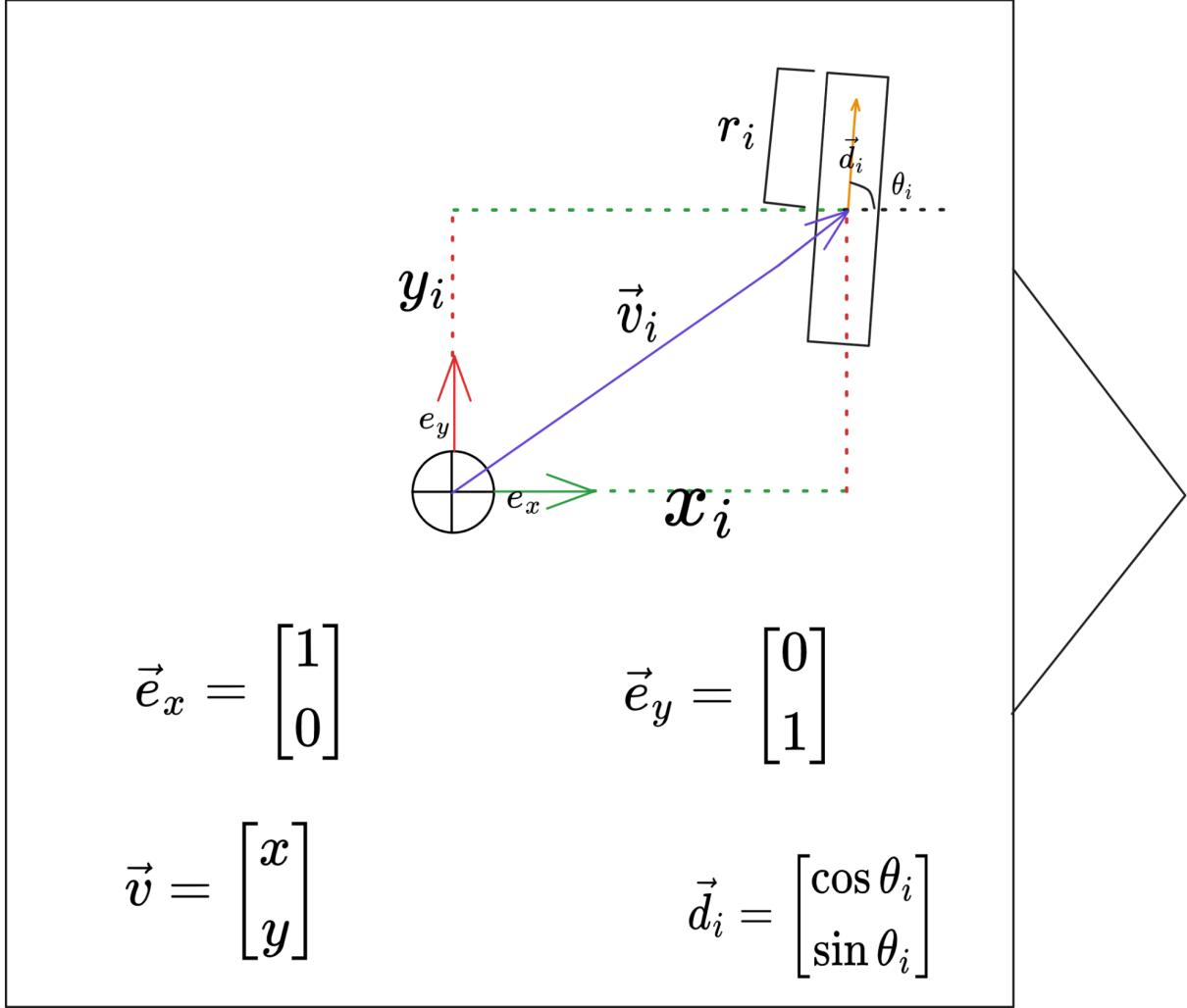


Figure 20: The configuration of a tracking wheel on the robot.  $(e_x, e_y)$  are the basis vectors of our coordinate system - the X and Y axes of the robot coordinate frame.  $d_i$  is the direction vector of the tracking wheel.

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  $\vec{v}$ .

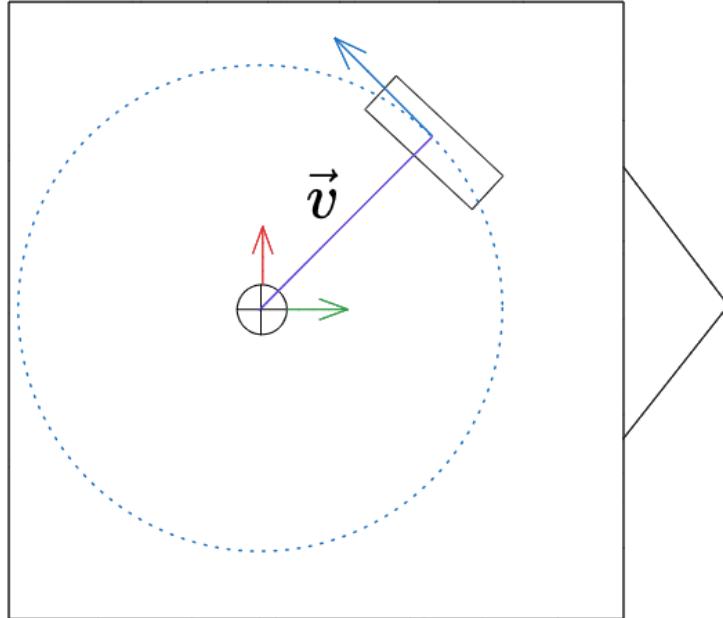
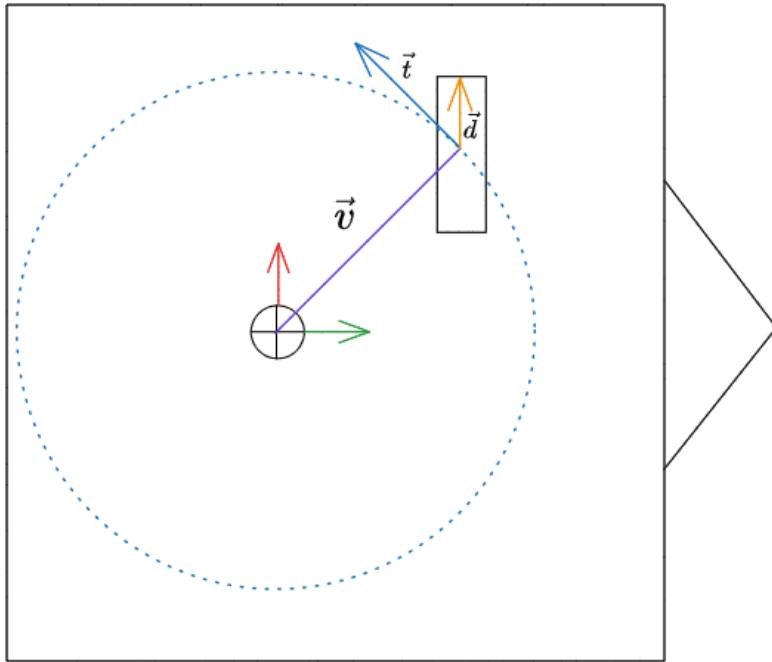


Figure 21: A conceptual perfectly perpendicular wheel

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.

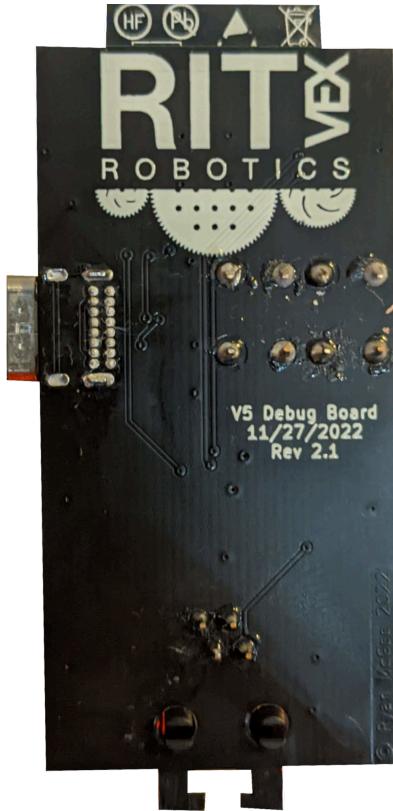


Figure 22 - Debug Board (Back)



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.

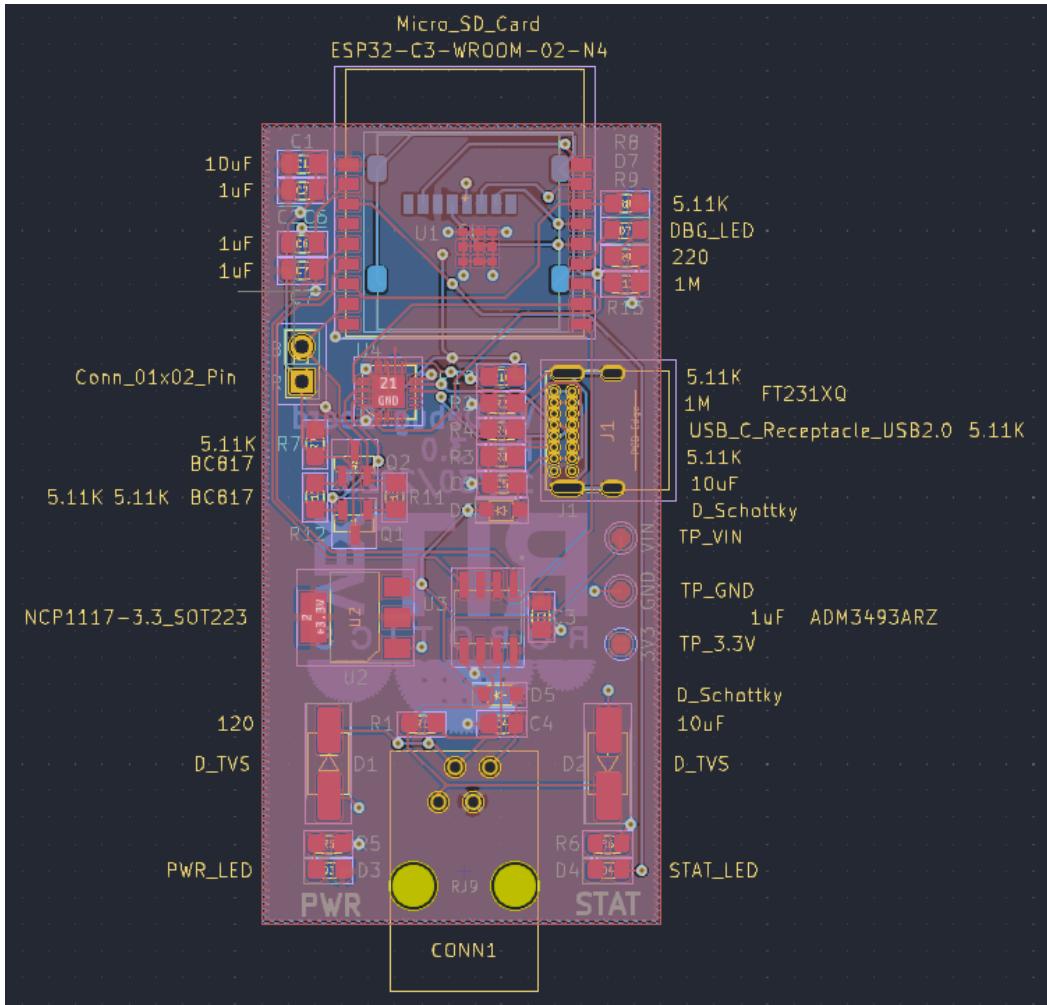


Figure 24 - Debug Board PCB Layout

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 Namespace Index</b>	<b>3</b>
2.1 Namespace List . . . . .	3
<b>3 Hierarchical Index</b>	<b>5</b>
3.1 Class Hierarchy . . . . .	5
<b>4 Class Index</b>	<b>9</b>
4.1 Class List . . . . .	9
<b>5 File Index</b>	<b>13</b>
5.1 File List . . . . .	13
<b>6 Namespace Documentation</b>	<b>15</b>
6.1 PurePursuit Namespace Reference . . . . .	15
6.1.1 Function Documentation . . . . .	15
6.1.1.1 estimate_remaining_dist() . . . . .	15
6.1.1.2 get_lookahead() . . . . .	16
6.1.1.3 inject_path() . . . . .	16
6.1.1.4 line_circle_intersections() . . . . .	16
6.1.1.5 smooth_path() . . . . .	16
6.1.1.6 smooth_path_cubic() . . . . .	17
6.1.1.7 smooth_path_hermite() . . . . .	17
6.2 screen Namespace Reference . . . . .	17
6.2.1 Typedef Documentation . . . . .	18
6.2.1.1 draw_func_t . . . . .	18
6.2.1.2 update_func_t . . . . .	18
6.2.2 Function Documentation . . . . .	18
6.2.2.1 draw_label() . . . . .	18
6.2.2.2 draw_widget() [1/2] . . . . .	19
6.2.2.3 draw_widget() [2/2] . . . . .	19
6.2.2.4 in_to_px() . . . . .	19
6.2.2.5 next_page() . . . . .	19
6.2.2.6 prev_page() . . . . .	19
6.2.2.7 start_screen() . . . . .	19
6.2.2.8 stop_screen() . . . . .	20
<b>7 Class Documentation</b>	<b>21</b>
7.1 AndCondition Class Reference . . . . .	21
7.1.1 Constructor & Destructor Documentation . . . . .	22
7.1.1.1 AndCondition() . . . . .	22

---

7.1.2 Member Function Documentation . . . . .	22
7.1.2.1 test() . . . . .	22
7.2 Async Class Reference . . . . .	22
7.2.1 Detailed Description . . . . .	23
7.2.2 Constructor & Destructor Documentation . . . . .	24
7.2.2.1 Async() . . . . .	24
7.2.3 Member Function Documentation . . . . .	24
7.2.3.1 run() . . . . .	24
7.3 AutoChooser Class Reference . . . . .	24
7.3.1 Detailed Description . . . . .	25
7.3.2 Constructor & Destructor Documentation . . . . .	26
7.3.2.1 AutoChooser() . . . . .	26
7.3.3 Member Function Documentation . . . . .	26
7.3.3.1 draw() . . . . .	26
7.3.3.2 get_choice() . . . . .	26
7.3.3.3 update() . . . . .	27
7.3.4 Member Data Documentation . . . . .	28
7.3.4.1 choice . . . . .	28
7.3.4.2 height . . . . .	28
7.3.4.3 list . . . . .	28
7.3.4.4 width . . . . .	28
7.4 AutoCommand Class Reference . . . . .	29
7.4.1 Member Function Documentation . . . . .	30
7.4.1.1 on_timeout() . . . . .	30
7.4.1.2 run() . . . . .	31
7.4.1.3 withCancelCondition() . . . . .	31
7.4.1.4 withTimeout() . . . . .	31
7.4.2 Member Data Documentation . . . . .	31
7.4.2.1 default_timeout . . . . .	31
7.4.2.2 timeout_seconds . . . . .	31
7.4.2.3 true_to_end . . . . .	32
7.5 BangBang Class Reference . . . . .	32
7.5.1 Constructor & Destructor Documentation . . . . .	33
7.5.1.1 BangBang() . . . . .	33
7.5.2 Member Function Documentation . . . . .	33
7.5.2.1 get() . . . . .	33
7.5.2.2 init() . . . . .	33
7.5.2.3 is_on_target() . . . . .	33
7.5.2.4 set_limits() . . . . .	34
7.5.2.5 update() . . . . .	34
7.6 BasicSolenoidSet Class Reference . . . . .	34
7.6.1 Detailed Description . . . . .	36

---

7.6.2 Constructor & Destructor Documentation . . . . .	36
7.6.2.1 BasicSolenoidSet() . . . . .	36
7.6.3 Member Function Documentation . . . . .	36
7.6.3.1 run() . . . . .	36
7.7 BasicSpinCommand Class Reference . . . . .	37
7.7.1 Detailed Description . . . . .	38
7.7.2 Member Enumeration Documentation . . . . .	38
7.7.2.1 type . . . . .	38
7.7.3 Constructor & Destructor Documentation . . . . .	38
7.7.3.1 BasicSpinCommand() . . . . .	38
7.7.4 Member Function Documentation . . . . .	39
7.7.4.1 run() . . . . .	39
7.8 BasicStopCommand Class Reference . . . . .	39
7.8.1 Detailed Description . . . . .	40
7.8.2 Constructor & Destructor Documentation . . . . .	41
7.8.2.1 BasicStopCommand() . . . . .	41
7.8.3 Member Function Documentation . . . . .	41
7.8.3.1 run() . . . . .	41
7.9 Branch Class Reference . . . . .	42
7.9.1 Detailed Description . . . . .	43
7.9.2 Constructor & Destructor Documentation . . . . .	43
7.9.2.1 Branch() . . . . .	43
7.9.2.2 ~Branch() . . . . .	43
7.9.3 Member Function Documentation . . . . .	43
7.9.3.1 on_timeout() . . . . .	43
7.9.3.2 run() . . . . .	44
7.10 screen::ButtonConfig Struct Reference . . . . .	44
7.10.1 Member Data Documentation . . . . .	44
7.10.1.1 onclick . . . . .	44
7.11 screen::ButtonWidget Class Reference . . . . .	44
7.11.1 Detailed Description . . . . .	45
7.11.2 Constructor & Destructor Documentation . . . . .	45
7.11.2.1 ButtonWidget() [1/2] . . . . .	45
7.11.2.2 ButtonWidget() [2/2] . . . . .	45
7.11.3 Member Function Documentation . . . . .	46
7.11.3.1 draw() . . . . .	46
7.11.3.2 update() . . . . .	46
7.12 screen::CheckboxConfig Struct Reference . . . . .	46
7.12.1 Member Data Documentation . . . . .	47
7.12.1.1 onupdate . . . . .	47
7.13 CommandController Class Reference . . . . .	47
7.13.1 Detailed Description . . . . .	47

7.13.2 Constructor & Destructor Documentation . . . . .	47
7.13.2.1 CommandController() [1/2] . . . . .	47
7.13.2.2 CommandController() [2/2] . . . . .	47
7.13.3 Member Function Documentation . . . . .	48
7.13.3.1 add() [1/3] . . . . .	48
7.13.3.2 add() [2/3] . . . . .	48
7.13.3.3 add() [3/3] . . . . .	48
7.13.3.4 add_cancel_func() . . . . .	50
7.13.3.5 add_delay() . . . . .	50
7.13.3.6 last_command_timed_out() . . . . .	50
7.13.3.7 run() . . . . .	51
7.14 Condition Class Reference . . . . .	51
7.14.1 Detailed Description . . . . .	51
7.14.2 Member Function Documentation . . . . .	51
7.14.2.1 And() . . . . .	51
7.14.2.2 Or() . . . . .	52
7.14.2.3 test() . . . . .	52
7.15 CustomEncoder Class Reference . . . . .	52
7.15.1 Detailed Description . . . . .	53
7.15.2 Constructor & Destructor Documentation . . . . .	53
7.15.2.1 CustomEncoder() . . . . .	53
7.15.3 Member Function Documentation . . . . .	53
7.15.3.1 position() . . . . .	53
7.15.3.2 rotation() . . . . .	54
7.15.3.3 setPosition() . . . . .	54
7.15.3.4 setRotation() . . . . .	54
7.15.3.5 velocity() . . . . .	54
7.16 DelayCommand Class Reference . . . . .	55
7.16.1 Detailed Description . . . . .	56
7.16.2 Constructor & Destructor Documentation . . . . .	56
7.16.2.1 DelayCommand() . . . . .	56
7.16.3 Member Function Documentation . . . . .	56
7.16.3.1 run() . . . . .	56
7.17 DriveForwardCommand Class Reference . . . . .	57
7.17.1 Detailed Description . . . . .	58
7.17.2 Constructor & Destructor Documentation . . . . .	58
7.17.2.1 DriveForwardCommand() . . . . .	58
7.17.3 Member Function Documentation . . . . .	59
7.17.3.1 on_timeout() . . . . .	59
7.17.3.2 run() . . . . .	59
7.18 DriveStopCommand Class Reference . . . . .	60
7.18.1 Detailed Description . . . . .	61

---

7.18.2 Constructor & Destructor Documentation . . . . .	61
7.18.2.1 DriveStopCommand() . . . . .	61
7.18.3 Member Function Documentation . . . . .	61
7.18.3.1 on_timeout() . . . . .	61
7.18.3.2 run() . . . . .	61
7.19 DriveToPointCommand Class Reference . . . . .	62
7.19.1 Detailed Description . . . . .	63
7.19.2 Constructor & Destructor Documentation . . . . .	63
7.19.2.1 DriveToPointCommand() [1/2] . . . . .	63
7.19.2.2 DriveToPointCommand() [2/2] . . . . .	64
7.19.3 Member Function Documentation . . . . .	64
7.19.3.1 run() . . . . .	64
7.20 AutoChooser::entry_t Struct Reference . . . . .	64
7.20.1 Detailed Description . . . . .	65
7.20.2 Member Data Documentation . . . . .	65
7.20.2.1 name . . . . .	65
7.20.2.2 rect . . . . .	65
7.21 ExponentialMovingAverage Class Reference . . . . .	66
7.21.1 Detailed Description . . . . .	66
7.21.2 Constructor & Destructor Documentation . . . . .	67
7.21.2.1 ExponentialMovingAverage() [1/2] . . . . .	67
7.21.2.2 ExponentialMovingAverage() [2/2] . . . . .	67
7.21.3 Member Function Documentation . . . . .	67
7.21.3.1 add_entry() . . . . .	67
7.21.3.2 get_size() . . . . .	67
7.21.3.3 get_value() . . . . .	68
7.22 Feedback Class Reference . . . . .	68
7.22.1 Detailed Description . . . . .	69
7.22.2 Member Function Documentation . . . . .	69
7.22.2.1 get() . . . . .	69
7.22.2.2 init() . . . . .	69
7.22.2.3 is_on_target() . . . . .	69
7.22.2.4 set_limits() . . . . .	70
7.22.2.5 update() . . . . .	70
7.23 FeedForward Class Reference . . . . .	70
7.23.1 Detailed Description . . . . .	71
7.23.2 Constructor & Destructor Documentation . . . . .	71
7.23.2.1 FeedForward() . . . . .	71
7.23.3 Member Function Documentation . . . . .	72
7.23.3.1 calculate() . . . . .	72
7.24 FeedForward:ff_config_t Struct Reference . . . . .	72
7.24.1 Detailed Description . . . . .	72

7.24.2 Member Data Documentation	73
7.24.2.1 kA	73
7.24.2.2 kG	73
7.24.2.3 kS	73
7.24.2.4 kV	73
7.25 Filter Class Reference	73
7.25.1 Detailed Description	74
7.25.2 Member Function Documentation	74
7.25.2.1 add_entry()	74
7.25.2.2 get_value()	74
7.26 Flywheel Class Reference	74
7.26.1 Detailed Description	75
7.26.2 Constructor & Destructor Documentation	75
7.26.2.1 Flywheel()	75
7.26.3 Member Function Documentation	75
7.26.3.1 get_motors()	75
7.26.3.2 get_target()	76
7.26.3.3 getRPM()	76
7.26.3.4 is_on_target()	76
7.26.3.5 Page()	76
7.26.3.6 spin_manual()	76
7.26.3.7 spin_rpm()	77
7.26.3.8 SpinRpmCmd()	77
7.26.3.9 stop()	78
7.26.3.10 WaitUntilUpToSpeedCmd()	78
7.26.4 Friends And Related Symbol Documentation	78
7.26.4.1 FlywheelPage	78
7.26.4.2 spinRPMTask	78
7.27 FlywheelPage Class Reference	79
7.27.1 Constructor & Destructor Documentation	79
7.27.1.1 FlywheelPage()	79
7.27.2 Member Function Documentation	80
7.27.2.1 draw()	80
7.27.2.2 update()	80
7.27.3 Member Data Documentation	80
7.27.3.1 window_size	80
7.28 FlywheelStopCommand Class Reference	81
7.28.1 Detailed Description	82
7.28.2 Constructor & Destructor Documentation	82
7.28.2.1 FlywheelStopCommand()	82
7.28.3 Member Function Documentation	82
7.28.3.1 run()	82

---

7.29 FlywheelStopMotorsCommand Class Reference . . . . .	83
7.29.1 Detailed Description . . . . .	84
7.29.2 Constructor & Destructor Documentation . . . . .	84
7.29.2.1 FlywheelStopMotorsCommand() . . . . .	84
7.29.3 Member Function Documentation . . . . .	84
7.29.3.1 run() . . . . .	84
7.30 FlywheelStopNonTasksCommand Class Reference . . . . .	85
7.30.1 Detailed Description . . . . .	86
7.31 FunctionCommand Class Reference . . . . .	86
7.31.1 Detailed Description . . . . .	87
7.31.2 Constructor & Destructor Documentation . . . . .	87
7.31.2.1 FunctionCommand() . . . . .	87
7.31.3 Member Function Documentation . . . . .	87
7.31.3.1 run() . . . . .	87
7.32 FunctionCondition Class Reference . . . . .	88
7.32.1 Detailed Description . . . . .	88
7.32.2 Constructor & Destructor Documentation . . . . .	89
7.32.2.1 FunctionCondition() . . . . .	89
7.32.3 Member Function Documentation . . . . .	89
7.32.3.1 test() . . . . .	89
7.33 screen::FunctionPage Class Reference . . . . .	89
7.33.1 Detailed Description . . . . .	90
7.33.2 Constructor & Destructor Documentation . . . . .	90
7.33.2.1 FunctionPage() . . . . .	90
7.33.3 Member Function Documentation . . . . .	91
7.33.3.1 draw() . . . . .	91
7.33.3.2 update() . . . . .	91
7.34 GenericAuto Class Reference . . . . .	91
7.34.1 Detailed Description . . . . .	92
7.34.2 Member Function Documentation . . . . .	92
7.34.2.1 add() . . . . .	92
7.34.2.2 add_async() . . . . .	92
7.34.2.3 add_delay() . . . . .	92
7.34.2.4 run() . . . . .	92
7.35 GraphDrawer Class Reference . . . . .	93
7.35.1 Constructor & Destructor Documentation . . . . .	93
7.35.1.1 GraphDrawer() . . . . .	93
7.35.2 Member Function Documentation . . . . .	94
7.35.2.1 add_samples() [1/2] . . . . .	94
7.35.2.2 add_samples() [2/2] . . . . .	94
7.35.2.3 draw() . . . . .	94
7.36 PurePursuit::hermite_point Struct Reference . . . . .	95

7.36.1 Detailed Description	95
7.36.2 Member Function Documentation	95
7.36.2.1 getPoint()	95
7.36.2.2 getTangent()	95
7.36.3 Member Data Documentation	95
7.36.3.1 dir	95
7.36.3.2 mag	95
7.36.3.3 x	96
7.36.3.4 y	96
7.37 IfTimePassed Class Reference	96
7.37.1 Detailed Description	97
7.37.2 Constructor & Destructor Documentation	97
7.37.2.1 IfTimePassed()	97
7.37.3 Member Function Documentation	97
7.37.3.1 test()	97
7.38 InOrder Class Reference	98
7.38.1 Detailed Description	99
7.38.2 Constructor & Destructor Documentation	99
7.38.2.1 InOrder() [1/3]	99
7.38.2.2 InOrder() [2/3]	99
7.38.2.3 InOrder() [3/3]	99
7.38.3 Member Function Documentation	100
7.38.3.1 on_timeout()	100
7.38.3.2 run()	100
7.39 screen::LabelConfig Struct Reference	100
7.39.1 Member Data Documentation	100
7.39.1.1 label	100
7.40 Lift< T > Class Template Reference	101
7.40.1 Detailed Description	101
7.40.2 Constructor & Destructor Documentation	101
7.40.2.1 Lift()	101
7.40.3 Member Function Documentation	102
7.40.3.1 control_continuous()	102
7.40.3.2 control_manual()	102
7.40.3.3 control_setpoints()	102
7.40.3.4 get_async()	103
7.40.3.5 get_setpoint()	103
7.40.3.6 hold()	103
7.40.3.7 home()	103
7.40.3.8 set_async()	103
7.40.3.9 set_position()	104
7.40.3.10 set_sensor_function()	104

---

7.40.3.11 set_sensor_reset() . . . . .	104
7.40.3.12 set_setpoint() . . . . .	104
7.41 Lift< T >::lift_cfg_t Struct Reference . . . . .	105
7.41.1 Detailed Description . . . . .	105
7.41.2 Member Data Documentation . . . . .	106
7.41.2.1 down_speed . . . . .	106
7.41.2.2 lift_pid_cfg . . . . .	106
7.41.2.3 softstop_down . . . . .	106
7.41.2.4 softstop_up . . . . .	106
7.41.2.5 up_speed . . . . .	106
7.42 Logger Class Reference . . . . .	106
7.42.1 Detailed Description . . . . .	107
7.42.2 Constructor & Destructor Documentation . . . . .	107
7.42.2.1 Logger() [1/2] . . . . .	107
7.42.2.2 Logger() [2/2] . . . . .	107
7.42.3 Member Function Documentation . . . . .	108
7.42.3.1 Log() [1/2] . . . . .	108
7.42.3.2 Log() [2/2] . . . . .	108
7.42.3.3 Logf() [1/2] . . . . .	108
7.42.3.4 Logf() [2/2] . . . . .	108
7.42.3.5 LogIn() [1/2] . . . . .	109
7.42.3.6 LogIn() [2/2] . . . . .	109
7.42.3.7 operator=() . . . . .	109
7.42.4 Member Data Documentation . . . . .	109
7.42.4.1 MAX_FORMAT_LEN . . . . .	109
7.43 MotionController::m_profile_cfg_t Struct Reference . . . . .	110
7.43.1 Detailed Description . . . . .	110
7.43.2 Member Data Documentation . . . . .	111
7.43.2.1 accel . . . . .	111
7.43.2.2 ff_cfg . . . . .	111
7.43.2.3 max_v . . . . .	111
7.43.2.4 pid_cfg . . . . .	111
7.44 Mat2 Struct Reference . . . . .	111
7.44.1 Member Function Documentation . . . . .	112
7.44.1.1 FromRotationDegrees() . . . . .	112
7.44.1.2 operator*() . . . . .	112
7.44.2 Member Data Documentation . . . . .	112
7.44.2.1 X11 . . . . .	112
7.44.2.2 X12 . . . . .	112
7.44.2.3 X21 . . . . .	112
7.44.2.4 X22 . . . . .	112
7.45 MecanumDrive Class Reference . . . . .	112

---

7.45.1 Detailed Description . . . . .	113
7.45.2 Constructor & Destructor Documentation . . . . .	113
7.45.2.1 MecanumDrive() . . . . .	113
7.45.3 Member Function Documentation . . . . .	113
7.45.3.1 auto_drive() . . . . .	113
7.45.3.2 auto_turn() . . . . .	114
7.45.3.3 drive() . . . . .	115
7.45.3.4 drive_raw() . . . . .	115
7.46 MecanumDrive::mecanumdrive_config_t Struct Reference . . . . .	116
7.46.1 Detailed Description . . . . .	116
7.46.2 Member Data Documentation . . . . .	116
7.46.2.1 drive_gyro_pid_conf . . . . .	116
7.46.2.2 drive_pid_conf . . . . .	116
7.46.2.3 drive_wheel_diam . . . . .	117
7.46.2.4 lateral_wheel_diam . . . . .	117
7.46.2.5 turn_pid_conf . . . . .	117
7.46.2.6 wheelbase_width . . . . .	117
7.47 motion_t Struct Reference . . . . .	117
7.47.1 Detailed Description . . . . .	117
7.47.2 Member Data Documentation . . . . .	117
7.47.2.1 accel . . . . .	117
7.47.2.2 pos . . . . .	118
7.47.2.3 vel . . . . .	118
7.48 MotionController Class Reference . . . . .	118
7.48.1 Detailed Description . . . . .	119
7.48.2 Constructor & Destructor Documentation . . . . .	119
7.48.2.1 MotionController() . . . . .	119
7.48.3 Member Function Documentation . . . . .	120
7.48.3.1 get() . . . . .	120
7.48.3.2 get_motion() . . . . .	120
7.48.3.3 init() . . . . .	120
7.48.3.4 is_on_target() . . . . .	121
7.48.3.5 Page() . . . . .	121
7.48.3.6 set_limits() . . . . .	121
7.48.3.7 tune_feedforward() . . . . .	121
7.48.3.8 update() . . . . .	122
7.48.4 Friends And Related Symbol Documentation . . . . .	122
7.48.4.1 MotionControllerPage . . . . .	122
7.49 MotionControllerPage Class Reference . . . . .	123
7.49.1 Constructor & Destructor Documentation . . . . .	123
7.49.1.1 MotionControllerPage() . . . . .	123
7.49.2 Member Function Documentation . . . . .	124

---

7.49.2.1 draw()	124
7.49.2.2 update()	124
7.50 MovingAverage Class Reference	124
7.50.1 Detailed Description	125
7.50.2 Constructor & Destructor Documentation	126
7.50.2.1 MovingAverage() [1/2]	126
7.50.2.2 MovingAverage() [2/2]	126
7.50.3 Member Function Documentation	126
7.50.3.1 add_entry()	126
7.50.3.2 get_size()	126
7.50.3.3 get_value()	127
7.51 Odometry3Wheel Class Reference	127
7.51.1 Detailed Description	129
7.51.2 Constructor & Destructor Documentation	129
7.51.2.1 Odometry3Wheel()	129
7.51.3 Member Function Documentation	130
7.51.3.1 tune()	130
7.51.3.2 update()	130
7.52 Odometry3Wheel::odometry3wheel_cfg_t Struct Reference	131
7.52.1 Detailed Description	131
7.52.2 Member Data Documentation	131
7.52.2.1 off_axis_center_dist	131
7.52.2.2 wheel_diam	131
7.52.2.3 wheelbase_dist	131
7.53 OdometryBase Class Reference	132
7.53.1 Detailed Description	133
7.53.2 Constructor & Destructor Documentation	133
7.53.2.1 OdometryBase()	133
7.53.3 Member Function Documentation	134
7.53.3.1 background_task()	134
7.53.3.2 end_async()	134
7.53.3.3 get_accel()	134
7.53.3.4 get-angular_accel_deg()	134
7.53.3.5 get-angular_speed_deg()	135
7.53.3.6 get_position()	135
7.53.3.7 get_speed()	135
7.53.3.8 pos_diff()	135
7.53.3.9 rot_diff()	136
7.53.3.10 set_position()	136
7.53.3.11 SetPositionCmd()	136
7.53.3.12 smallest_angle()	136
7.53.3.13 update()	137

7.53.4 Member Data Documentation	137
7.53.4.1 accel	137
7.53.4.2 ang_accel_deg	137
7.53.4.3 ang_speed_deg	137
7.53.4.4 current_pos	138
7.53.4.5 end_task	138
7.53.4.6 handle	138
7.53.4.7 mut	138
7.53.4.8 speed	138
7.53.4.9 zero_pos	138
7.54 screen::OdometryPage Class Reference	139
7.54.1 Detailed Description	139
7.54.2 Constructor & Destructor Documentation	140
7.54.2.1 OdometryPage()	140
7.54.3 Member Function Documentation	140
7.54.3.1 draw()	140
7.54.3.2 update()	140
7.55 OdometryTank Class Reference	141
7.55.1 Detailed Description	142
7.55.2 Constructor & Destructor Documentation	142
7.55.2.1 OdometryTank() [1/3]	142
7.55.2.2 OdometryTank() [2/3]	143
7.55.2.3 OdometryTank() [3/3]	143
7.55.3 Member Function Documentation	144
7.55.3.1 set_position()	144
7.55.3.2 update()	144
7.56 OdomSetPosition Class Reference	144
7.56.1 Detailed Description	145
7.56.2 Constructor & Destructor Documentation	146
7.56.2.1 OdomSetPosition()	146
7.56.3 Member Function Documentation	147
7.56.3.1 run()	147
7.57 OrCondition Class Reference	147
7.57.1 Constructor & Destructor Documentation	148
7.57.1.1 OrCondition()	148
7.57.2 Member Function Documentation	148
7.57.2.1 test()	148
7.58 screen::Page Class Reference	149
7.58.1 Detailed Description	149
7.58.2 Member Function Documentation	150
7.58.2.1 draw()	150
7.58.2.2 update()	150

---

7.59 Parallel Class Reference . . . . .	150
7.59.1 Detailed Description . . . . .	152
7.59.2 Constructor & Destructor Documentation . . . . .	152
7.59.2.1 Parallel() . . . . .	152
7.59.3 Member Function Documentation . . . . .	152
7.59.3.1 on_timeout() . . . . .	152
7.59.3.2 run() . . . . .	152
7.60 parallel_runner_info Struct Reference . . . . .	153
7.60.1 Member Data Documentation . . . . .	153
7.60.1.1 cmd . . . . .	153
7.60.1.2 index . . . . .	153
7.60.1.3 runners . . . . .	153
7.61 PurePursuit::Path Class Reference . . . . .	154
7.61.1 Detailed Description . . . . .	154
7.61.2 Constructor & Destructor Documentation . . . . .	154
7.61.2.1 Path() . . . . .	154
7.61.3 Member Function Documentation . . . . .	154
7.61.3.1 get_points() . . . . .	154
7.61.3.2 get_radius() . . . . .	154
7.61.3.3 is_valid() . . . . .	155
7.62 PID Class Reference . . . . .	155
7.62.1 Detailed Description . . . . .	156
7.62.2 Member Enumeration Documentation . . . . .	156
7.62.2.1 ERROR_TYPE . . . . .	156
7.62.3 Constructor & Destructor Documentation . . . . .	157
7.62.3.1 PID() . . . . .	157
7.62.4 Member Function Documentation . . . . .	157
7.62.4.1 get() . . . . .	157
7.62.4.2 get_error() . . . . .	157
7.62.4.3 get_sensor_val() . . . . .	158
7.62.4.4 get_target() . . . . .	158
7.62.4.5 init() . . . . .	158
7.62.4.6 is_on_target() . . . . .	158
7.62.4.7 reset() . . . . .	159
7.62.4.8 set_limits() . . . . .	159
7.62.4.9 set_target() . . . . .	159
7.62.4.10 update() . . . . .	159
7.62.5 Member Data Documentation . . . . .	160
7.62.5.1 config . . . . .	160
7.63 PID::pid_config_t Struct Reference . . . . .	160
7.63.1 Detailed Description . . . . .	161
7.63.2 Member Data Documentation . . . . .	161

---

7.63.2.1 d . . . . .	161
7.63.2.2 deadband . . . . .	161
7.63.2.3 error_method . . . . .	161
7.63.2.4 i . . . . .	161
7.63.2.5 on_target_time . . . . .	161
7.63.2.6 p . . . . .	162
7.64 PIDFF Class Reference . . . . .	162
7.64.1 Constructor & Destructor Documentation . . . . .	163
7.64.1.1 PIDFF() . . . . .	163
7.64.2 Member Function Documentation . . . . .	163
7.64.2.1 get() . . . . .	163
7.64.2.2 get_sensor_val() . . . . .	163
7.64.2.3 get_target() . . . . .	163
7.64.2.4 init() . . . . .	163
7.64.2.5 is_on_target() . . . . .	164
7.64.2.6 reset() . . . . .	164
7.64.2.7 set_limits() . . . . .	164
7.64.2.8 set_target() . . . . .	165
7.64.2.9 update() [1/2] . . . . .	165
7.64.2.10 update() [2/2] . . . . .	165
7.64.3 Member Data Documentation . . . . .	166
7.64.3.1 pid . . . . .	166
7.65 screen::PIDPage Class Reference . . . . .	166
7.65.1 Detailed Description . . . . .	167
7.65.2 Constructor & Destructor Documentation . . . . .	167
7.65.2.1 PIDPage() [1/2] . . . . .	167
7.65.2.2 PIDPage() [2/2] . . . . .	167
7.65.3 Member Function Documentation . . . . .	167
7.65.3.1 draw() . . . . .	167
7.65.3.2 update() . . . . .	168
7.66 point_t Struct Reference . . . . .	168
7.66.1 Detailed Description . . . . .	168
7.66.2 Member Function Documentation . . . . .	168
7.66.2.1 dist() . . . . .	168
7.66.2.2 operator*() . . . . .	169
7.66.2.3 operator+() [1/2] . . . . .	169
7.66.2.4 operator+() [2/2] . . . . .	169
7.66.2.5 operator-() [1/2] . . . . .	169
7.66.2.6 operator-() [2/2] . . . . .	169
7.66.2.7 operator/() . . . . .	170
7.66.2.8 operator==( ) . . . . .	170
7.66.3 Member Data Documentation . . . . .	170

---

7.66.3.1 x . . . . .	170
7.66.3.2 y . . . . .	170
7.67 pose_t Struct Reference . . . . .	170
7.67.1 Detailed Description . . . . .	171
7.67.2 Member Function Documentation . . . . .	171
7.67.2.1 get_point() . . . . .	171
7.67.3 Member Data Documentation . . . . .	171
7.67.3.1 rot . . . . .	171
7.67.3.2 x . . . . .	171
7.67.3.3 y . . . . .	171
7.68 PurePursuitCommand Class Reference . . . . .	172
7.68.1 Detailed Description . . . . .	173
7.68.2 Constructor & Destructor Documentation . . . . .	173
7.68.2.1 PurePursuitCommand() . . . . .	173
7.68.3 Member Function Documentation . . . . .	173
7.68.3.1 on_timeout() . . . . .	173
7.68.3.2 run() . . . . .	174
7.69 Rect Struct Reference . . . . .	174
7.69.1 Member Function Documentation . . . . .	175
7.69.1.1 center() . . . . .	175
7.69.1.2 contains() . . . . .	175
7.69.1.3 dimensions() . . . . .	175
7.69.1.4 from_min_and_size() . . . . .	175
7.69.1.5 height() . . . . .	175
7.69.1.6 width() . . . . .	175
7.69.2 Member Data Documentation . . . . .	175
7.69.2.1 max . . . . .	175
7.69.2.2 min . . . . .	175
7.70 RepeatUntil Class Reference . . . . .	176
7.70.1 Constructor & Destructor Documentation . . . . .	177
7.70.1.1 RepeatUntil() [1/2] . . . . .	177
7.70.1.2 RepeatUntil() [2/2] . . . . .	177
7.70.2 Member Function Documentation . . . . .	178
7.70.2.1 on_timeout() . . . . .	178
7.70.2.2 run() . . . . .	178
7.71 robot_specs_t Struct Reference . . . . .	178
7.71.1 Detailed Description . . . . .	179
7.71.2 Member Data Documentation . . . . .	179
7.71.2.1 correction_pid . . . . .	179
7.71.2.2 dist_between_wheels . . . . .	179
7.71.2.3 drive_correction_cutoff . . . . .	179
7.71.2.4 drive_feedback . . . . .	180

7.71.2.5 odom_gear_ratio . . . . .	180
7.71.2.6 odom_wheel_diam . . . . .	180
7.71.2.7 robot_radius . . . . .	180
7.71.2.8 turn_feedback . . . . .	180
7.72 screen::ScreenData Struct Reference . . . . .	180
7.72.1 Detailed Description . . . . .	181
7.72.2 Constructor & Destructor Documentation . . . . .	181
7.72.2.1 ScreenData() . . . . .	181
7.72.3 Member Data Documentation . . . . .	181
7.72.3.1 page . . . . .	181
7.72.3.2 pages . . . . .	181
7.72.3.3 screen . . . . .	181
7.73 screen::ScreenRect Struct Reference . . . . .	181
7.73.1 Member Data Documentation . . . . .	182
7.73.1.1 x1 . . . . .	182
7.73.1.2 x2 . . . . .	182
7.73.1.3 y1 . . . . .	182
7.73.1.4 y2 . . . . .	182
7.74 Serializer Class Reference . . . . .	182
7.74.1 Detailed Description . . . . .	183
7.74.2 Constructor & Destructor Documentation . . . . .	183
7.74.2.1 ~Serializer() . . . . .	183
7.74.2.2 Serializer() . . . . .	183
7.74.3 Member Function Documentation . . . . .	183
7.74.3.1 bool_or() . . . . .	183
7.74.3.2 double_or() . . . . .	184
7.74.3.3 int_or() . . . . .	184
7.74.3.4 save_to_disk() . . . . .	184
7.74.3.5 set_bool() . . . . .	184
7.74.3.6 set_double() . . . . .	185
7.74.3.7 set_int() . . . . .	185
7.74.3.8 set_string() . . . . .	185
7.74.3.9 string_or() . . . . .	186
7.75 screen::SizedWidget Struct Reference . . . . .	186
7.75.1 Member Data Documentation . . . . .	187
7.75.1.1 size . . . . .	187
7.75.1.2 widget . . . . .	187
7.76 SliderCfg Struct Reference . . . . .	187
7.76.1 Member Data Documentation . . . . .	187
7.76.1.1 max . . . . .	187
7.76.1.2 min . . . . .	187
7.76.1.3 val . . . . .	187

---

7.77 screen::SliderConfig Struct Reference . . . . .	188
7.77.1 Member Data Documentation . . . . .	188
7.77.1.1 high . . . . .	188
7.77.1.2 low . . . . .	188
7.77.1.3 val . . . . .	188
7.78 screen::SliderWidget Class Reference . . . . .	188
7.78.1 Detailed Description . . . . .	189
7.78.2 Constructor & Destructor Documentation . . . . .	189
7.78.2.1 SliderWidget() . . . . .	189
7.78.3 Member Function Documentation . . . . .	189
7.78.3.1 draw() . . . . .	189
7.78.3.2 update() . . . . .	189
7.79 SpinRPMCommand Class Reference . . . . .	190
7.79.1 Detailed Description . . . . .	191
7.79.2 Constructor & Destructor Documentation . . . . .	191
7.79.2.1 SpinRPMCommand() . . . . .	191
7.79.3 Member Function Documentation . . . . .	192
7.79.3.1 run() . . . . .	192
7.80 PurePursuit::spline Struct Reference . . . . .	192
7.80.1 Detailed Description . . . . .	192
7.80.2 Member Function Documentation . . . . .	192
7.80.2.1 getY() . . . . .	192
7.80.3 Member Data Documentation . . . . .	193
7.80.3.1 a . . . . .	193
7.80.3.2 b . . . . .	193
7.80.3.3 c . . . . .	193
7.80.3.4 d . . . . .	193
7.80.3.5 x_end . . . . .	193
7.80.3.6 x_start . . . . .	193
7.81 screen::StatsPage Class Reference . . . . .	194
7.81.1 Detailed Description . . . . .	194
7.81.2 Constructor & Destructor Documentation . . . . .	195
7.81.2.1 StatsPage() . . . . .	195
7.81.3 Member Function Documentation . . . . .	196
7.81.3.1 draw() . . . . .	196
7.81.3.2 update() . . . . .	196
7.82 TakeBackHalf Class Reference . . . . .	197
7.82.1 Detailed Description . . . . .	198
7.82.2 Constructor & Destructor Documentation . . . . .	198
7.82.2.1 TakeBackHalf() . . . . .	198
7.82.3 Member Function Documentation . . . . .	198
7.82.3.1 get() . . . . .	198

---

7.82.3.2 init()	198
7.82.3.3 is_on_target()	199
7.82.3.4 set_limits()	199
7.82.3.5 update()	199
7.82.4 Member Data Documentation	199
7.82.4.1 first_cross_split	199
7.82.4.2 TBH_gain	200
7.83 TankDrive Class Reference	200
7.83.1 Detailed Description	201
7.83.2 Member Enumeration Documentation	201
7.83.2.1 BrakeType	201
7.83.3 Constructor & Destructor Documentation	201
7.83.3.1 TankDrive()	201
7.83.4 Member Function Documentation	202
7.83.4.1 drive_arcade()	202
7.83.4.2 drive_forward() [1/2]	202
7.83.4.3 drive_forward() [2/2]	203
7.83.4.4 drive_tank()	204
7.83.4.5 drive_tank_raw()	204
7.83.4.6 drive_to_point() [1/2]	204
7.83.4.7 drive_to_point() [2/2]	205
7.83.4.8 DriveForwardCmd() [1/2]	206
7.83.4.9 DriveForwardCmd() [2/2]	206
7.83.4.10 DriveToPointCmd() [1/2]	206
7.83.4.11 DriveToPointCmd() [2/2]	206
7.83.4.12 modify_inputs()	207
7.83.4.13 pure_pursuit() [1/2]	208
7.83.4.14 pure_pursuit() [2/2]	209
7.83.4.15 PurePursuitCmd() [1/2]	210
7.83.4.16 PurePursuitCmd() [2/2]	210
7.83.4.17 reset_auto()	210
7.83.4.18 stop()	210
7.83.4.19 turn_degrees() [1/2]	210
7.83.4.20 turn_degrees() [2/2]	211
7.83.4.21 turn_to_heading() [1/2]	212
7.83.4.22 turn_to_heading() [2/2]	212
7.83.4.23 TurnDegreesCmd() [1/2]	213
7.83.4.24 TurnDegreesCmd() [2/2]	213
7.83.4.25 TurnToHeadingCmd() [1/2]	213
7.83.4.26 TurnToHeadingCmd() [2/2]	213
7.84 screen::TextConfig Struct Reference	214
7.84.1 Member Data Documentation	214

---

7.84.1.1 <code>text</code>	214
7.85 <code>TimesTestedCondition</code> Class Reference	214
7.85.1 Constructor & Destructor Documentation	215
7.85.1.1 <code>TimesTestedCondition()</code>	215
7.85.2 Member Function Documentation	215
7.85.2.1 <code>test()</code>	215
7.86 <code>trapezoid_profile_segment_t</code> Struct Reference	215
7.86.1 Detailed Description	216
7.86.2 Member Data Documentation	216
7.86.2.1 <code>accel</code>	216
7.86.2.2 <code>duration</code>	216
7.86.2.3 <code>pos_after</code>	216
7.86.2.4 <code>vel_after</code>	216
7.87 <code>TrapezoidProfile</code> Class Reference	216
7.87.1 Detailed Description	217
7.87.2 Constructor & Destructor Documentation	218
7.87.2.1 <code>TrapezoidProfile()</code>	218
7.87.3 Member Function Documentation	218
7.87.3.1 <code>calculate()</code>	218
7.87.3.2 <code>calculate_time_based()</code>	218
7.87.3.3 <code>get_accel()</code>	219
7.87.3.4 <code>get_max_v()</code>	219
7.87.3.5 <code>get_movement_time()</code>	219
7.87.3.6 <code>set_accel()</code>	219
7.87.3.7 <code>set_endpts()</code>	219
7.87.3.8 <code>set_max_v()</code>	220
7.87.3.9 <code>set_vel_endpts()</code>	220
7.88 <code>TurnDegreesCommand</code> Class Reference	220
7.88.1 Detailed Description	221
7.88.2 Constructor & Destructor Documentation	222
7.88.2.1 <code>TurnDegreesCommand()</code>	222
7.88.3 Member Function Documentation	222
7.88.3.1 <code>on_timeout()</code>	222
7.88.3.2 <code>run()</code>	222
7.89 <code>TurnToHeadingCommand</code> Class Reference	223
7.89.1 Detailed Description	224
7.89.2 Constructor & Destructor Documentation	224
7.89.2.1 <code>TurnToHeadingCommand()</code>	224
7.89.3 Member Function Documentation	224
7.89.3.1 <code>on_timeout()</code>	224
7.89.3.2 <code>run()</code>	225
7.90 <code>Vector2D</code> Class Reference	225

---

7.90.1 Detailed Description . . . . .	225
7.90.2 Constructor & Destructor Documentation . . . . .	225
7.90.2.1 Vector2D() [1/2] . . . . .	225
7.90.2.2 Vector2D() [2/2] . . . . .	226
7.90.3 Member Function Documentation . . . . .	226
7.90.3.1 get_dir() . . . . .	226
7.90.3.2 get_mag() . . . . .	226
7.90.3.3 get_x() . . . . .	227
7.90.3.4 get_y() . . . . .	227
7.90.3.5 normalize() . . . . .	227
7.90.3.6 operator*() . . . . .	227
7.90.3.7 operator+() . . . . .	228
7.90.3.8 operator-() . . . . .	228
7.90.3.9 point() . . . . .	228
7.91 WaitUntilCondition Class Reference . . . . .	229
7.91.1 Detailed Description . . . . .	230
7.91.2 Constructor & Destructor Documentation . . . . .	230
7.91.2.1 WaitUntilCondition() . . . . .	230
7.91.3 Member Function Documentation . . . . .	230
7.91.3.1 run() . . . . .	230
7.92 WaitUntilUpToSpeedCommand Class Reference . . . . .	231
7.92.1 Detailed Description . . . . .	232
7.92.2 Constructor & Destructor Documentation . . . . .	232
7.92.2.1 WaitUntilUpToSpeedCommand() . . . . .	232
7.92.3 Member Function Documentation . . . . .	232
7.92.3.1 run() . . . . .	232
7.93 screen::WidgetConfig Struct Reference . . . . .	233
7.93.1 Member Enumeration Documentation . . . . .	233
7.93.1.1 Type . . . . .	233
7.93.2 Member Data Documentation . . . . .	234
7.93.2.1 button . . . . .	234
7.93.2.2 checkbox . . . . .	234
7.93.2.3 [union] . . . . .	234
7.93.2.4 graph . . . . .	234
7.93.2.5 label . . . . .	234
7.93.2.6 slider . . . . .	234
7.93.2.7 text . . . . .	234
7.93.2.8 type . . . . .	235
7.93.2.9 widgets . . . . .	235
7.94 screen::WidgetPage Class Reference . . . . .	235
7.94.1 Constructor & Destructor Documentation . . . . .	236
7.94.1.1 WidgetPage() . . . . .	236

---

7.94.2 Member Function Documentation . . . . .	236
7.94.2.1 draw() . . . . .	236
7.94.2.2 update() . . . . .	236
<b>8 File Documentation</b> . . . . .	<b>239</b>
8.1 include/robot_specs.h File Reference . . . . .	239
8.2 robot_specs.h . . . . .	239
8.3 include/subsystems/custom_encoder.h File Reference . . . . .	240
8.4 custom_encoder.h . . . . .	240
8.5 include/subsystems/flywheel.h File Reference . . . . .	241
8.6 flywheel.h . . . . .	241
8.7 include/subsystems/layout.h File Reference . . . . .	242
8.8 layout.h . . . . .	242
8.9 include/subsystems/lift.h File Reference . . . . .	243
8.10 lift.h . . . . .	243
8.11 include/subsystems/mecanum_drive.h File Reference . . . . .	246
8.11.1 Macro Definition Documentation . . . . .	247
8.11.1.1 PI . . . . .	247
8.12 mecanum_drive.h . . . . .	247
8.13 include/subsystems/odometry/odometry_3wheel.h File Reference . . . . .	248
8.14 odometry_3wheel.h . . . . .	248
8.15 include/subsystems/odometry/odometry_base.h File Reference . . . . .	249
8.15.1 Macro Definition Documentation . . . . .	249
8.15.1.1 PI . . . . .	249
8.16 odometry_base.h . . . . .	250
8.17 include/subsystems/odometry/odometry_tank.h File Reference . . . . .	250
8.18 odometry_tank.h . . . . .	251
8.19 include/subsystems/screen.h File Reference . . . . .	251
8.20 screen.h . . . . .	253
8.21 include/subsystems/tank_drive.h File Reference . . . . .	256
8.21.1 Macro Definition Documentation . . . . .	256
8.21.1.1 PI . . . . .	256
8.22 tank_drive.h . . . . .	256
8.23 include/utils/auto_chooser.h File Reference . . . . .	258
8.24 auto_chooser.h . . . . .	258
8.25 include/utils/command_structure/auto_command.h File Reference . . . . .	259
8.26 auto_command.h . . . . .	260
8.27 include/utils/command_structure/basic_command.h File Reference . . . . .	262
8.28 basic_command.h . . . . .	262
8.29 include/utils/command_structure/command_controller.h File Reference . . . . .	263
8.30 command_controller.h . . . . .	264
8.31 include/utils/command_structure/delay_command.h File Reference . . . . .	265

8.32 delay_command.h . . . . .	265
8.33 include/utils/command_structure/drive_commands.h File Reference . . . . .	265
8.34 drive_commands.h . . . . .	266
8.35 include/utils/command_structure/flywheel_commands.h File Reference . . . . .	268
8.36 flywheel_commands.h . . . . .	269
8.37 include/utils/controls/bang_bang.h File Reference . . . . .	270
8.38 bang_bang.h . . . . .	270
8.39 include/utils/controls/feedback_base.h File Reference . . . . .	270
8.40 feedback_base.h . . . . .	271
8.41 include/utils/controls/feedforward.h File Reference . . . . .	271
8.41.1 Function Documentation . . . . .	272
8.41.1.1 tune_feedforward() . . . . .	272
8.42 feedforward.h . . . . .	273
8.43 include/utils/controls/motion_controller.h File Reference . . . . .	273
8.44 motion_controller.h . . . . .	274
8.45 include/utils/controls/pid.h File Reference . . . . .	275
8.46 pid.h . . . . .	275
8.47 include/utils/controls/pidff.h File Reference . . . . .	276
8.48 pidff.h . . . . .	277
8.49 include/utils/controls/take_back_half.h File Reference . . . . .	277
8.50 take_back_half.h . . . . .	278
8.51 include/utils/controls/trapezoid_profile.h File Reference . . . . .	278
8.51.1 Variable Documentation . . . . .	278
8.51.1.1 MAX_TRAPEZOID_PROFILE_SEGMENTS . . . . .	278
8.52 trapezoid_profile.h . . . . .	279
8.53 include/utils/generic_auto.h File Reference . . . . .	279
8.53.1 Typedef Documentation . . . . .	280
8.53.1.1 state_ptr . . . . .	280
8.54 generic_auto.h . . . . .	280
8.55 include/utils/geometry.h File Reference . . . . .	281
8.56 geometry.h . . . . .	281
8.57 include/utils/graph_drawer.h File Reference . . . . .	283
8.58 graph_drawer.h . . . . .	283
8.59 include/utils/logger.h File Reference . . . . .	284
8.59.1 Enumeration Type Documentation . . . . .	284
8.59.1.1 LogLevel . . . . .	284
8.60 logger.h . . . . .	285
8.61 include/utils/math_util.h File Reference . . . . .	285
8.61.1 Function Documentation . . . . .	286
8.61.1.1 calculate_linear_regression() . . . . .	286
8.61.1.2 clamp() . . . . .	286
8.61.1.3 covariance() . . . . .	286

---

8.61.1.4 estimate_path_length() . . . . .	286
8.61.1.5 lerp() . . . . .	287
8.61.1.6 mean() . . . . .	287
8.61.1.7 sign() . . . . .	287
8.61.1.8 variance() . . . . .	287
8.61.1.9 wrap_angle_deg() . . . . .	288
8.61.1.10 wrap_angle_rad() . . . . .	288
8.62 math_util.h . . . . .	288
8.63 include/utils/moving_average.h File Reference . . . . .	289
8.64 moving_average.h . . . . .	289
8.65 include/utils/pure_pursuit.h File Reference . . . . .	290
8.66 pure_pursuit.h . . . . .	291
8.67 include/utils/serializer.h File Reference . . . . .	292
8.67.1 Variable Documentation . . . . .	293
8.67.1.1 MAX_FILE_SIZE . . . . .	293
8.67.1.2 serialization_separator . . . . .	293
8.68 serializer.h . . . . .	294
8.69 include/utils/vector2d.h File Reference . . . . .	294
8.69.1 Macro Definition Documentation . . . . .	295
8.69.1.1 PI . . . . .	295
8.69.2 Function Documentation . . . . .	295
8.69.2.1 deg2rad() . . . . .	295
8.69.2.2 rad2deg() . . . . .	296
8.70 vector2d.h . . . . .	296
8.71 README.md File Reference . . . . .	297
8.72 src/subsystems/custom_encoder.cpp File Reference . . . . .	297
8.73 src/subsystems/flywheel.cpp File Reference . . . . .	297
8.73.1 Function Documentation . . . . .	298
8.73.1.1 spinRPMTask() . . . . .	298
8.74 src/subsystems/mecanum_drive.cpp File Reference . . . . .	298
8.75 src/subsystems/odometry/odometry_3wheel.cpp File Reference . . . . .	298
8.76 src/subsystems/odometry/odometry_base.cpp File Reference . . . . .	299
8.77 src/subsystems/odometry/odometry_tank.cpp File Reference . . . . .	299
8.78 src/subsystems/screen.cpp File Reference . . . . .	299
8.79 src/subsystems/tank_drive.cpp File Reference . . . . .	300
8.79.1 Variable Documentation . . . . .	301
8.79.1.1 captured_position . . . . .	301
8.79.1.2 was_breaking . . . . .	301
8.80 src/utils/auto_chooser.cpp File Reference . . . . .	301
8.81 src/utils/command_structure/auto_command.cpp File Reference . . . . .	302
8.82 src/utils/command_structure/basic_command.cpp File Reference . . . . .	302
8.83 src/utils/command_structure/command_controller.cpp File Reference . . . . .	303

---

8.84 src/utils/command_structure/drive_commands.cpp File Reference . . . . .	303
8.85 src/utils/command_structure/flywheel_commands.cpp File Reference . . . . .	303
8.86 src/utils/controls/bang_bang.cpp File Reference . . . . .	304
8.87 src/utils/controls/feedforward.cpp File Reference . . . . .	304
8.87.1 Function Documentation . . . . .	305
8.87.1.1 tune_feedforward() . . . . .	305
8.88 src/utils/controls/motion_controller.cpp File Reference . . . . .	305
8.89 src/utils/controls/pid.cpp File Reference . . . . .	306
8.90 src/utils/controls/pidff.cpp File Reference . . . . .	306
8.91 src/utils/controls/take_back_half.cpp File Reference . . . . .	307
8.92 src/utils/generic_auto.cpp File Reference . . . . .	307
8.93 src/utils/graph_drawer.cpp File Reference . . . . .	308
8.94 src/utils/logger.cpp File Reference . . . . .	308
8.95 src/utils/math_util.cpp File Reference . . . . .	309
8.95.1 Macro Definition Documentation . . . . .	310
8.95.1.1 PI . . . . .	310
8.95.2 Function Documentation . . . . .	310
8.95.2.1 calculate_linear_regression() . . . . .	310
8.95.2.2 clamp() . . . . .	310
8.95.2.3 covariance() . . . . .	310
8.95.2.4 estimate_path_length() . . . . .	310
8.95.2.5 lerp() . . . . .	311
8.95.2.6 mean() . . . . .	311
8.95.2.7 sign() . . . . .	311
8.95.2.8 variance() . . . . .	311
8.95.2.9 wrap_angle_deg() . . . . .	311
8.95.2.10 wrap_angle_rad() . . . . .	312
8.96 src/utils/moving_average.cpp File Reference . . . . .	312
8.97 src/utils/pure_pursuit.cpp File Reference . . . . .	312
8.98 src/utils/serializer.cpp File Reference . . . . .	313
8.98.1 Function Documentation . . . . .	313
8.98.1.1 from_bytes() [1/2] . . . . .	313
8.98.1.2 from_bytes() [2/2] . . . . .	314
8.98.1.3 sanitize_name() . . . . .	314
8.98.1.4 to_bytes() . . . . .	314
8.98.1.5 to_bytes< std::string >() . . . . .	314
8.99 src/utils/trapezoid_profile.cpp File Reference . . . . .	314
8.99.1 Function Documentation . . . . .	315
8.99.1.1 calc_pos() . . . . .	315
8.99.1.2 calc_vel() . . . . .	315
8.99.2 Variable Documentation . . . . .	315
8.99.2.1 EPSILON . . . . .	315

---

8.100 src/utils/vector2d.cpp File Reference . . . . .	316
8.100.1 Function Documentation . . . . .	316
8.100.1.1 deg2rad() . . . . .	316
8.100.1.2 rad2deg() . . . . .	316
<b>Index</b>	<b>317</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

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

PurePursuit . . . . .	15
screen . . . . .	17



# Chapter 3

## Hierarchical Index

### 3.1 Class Hierarchy

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

AutoCommand . . . . .	29
Async . . . . .	22
BasicSolenoidSet . . . . .	34
BasicSpinCommand . . . . .	37
BasicStopCommand . . . . .	39
Branch . . . . .	42
DelayCommand . . . . .	55
DriveForwardCommand . . . . .	57
DriveStopCommand . . . . .	60
DriveToPointCommand . . . . .	62
FlywheelStopCommand . . . . .	81
FlywheelStopMotorsCommand . . . . .	83
FlywheelStopNonTasksCommand . . . . .	85
FunctionCommand . . . . .	86
InOrder . . . . .	98
OdomSetPosition . . . . .	144
Parallel . . . . .	150
PurePursuitCommand . . . . .	172
RepeatUntil . . . . .	176
SpinRPMCommand . . . . .	190
TurnDegreesCommand . . . . .	220
TurnToHeadingCommand . . . . .	223
WaitUntilCondition . . . . .	229
WaitUntilUpToSpeedCommand . . . . .	231
screen::ButtonConfig . . . . .	44
screen::ButtonWidget . . . . .	44
screen::CheckboxConfig . . . . .	46
CommandController . . . . .	47
Condition . . . . .	51
AndCondition . . . . .	21
FunctionCondition . . . . .	88
IfTimePassed . . . . .	96
OrCondition . . . . .	147
TimesTestedCondition . . . . .	214
vex::encoder	

CustomEncoder . . . . .	52
AutoChooser::entry_t . . . . .	64
Feedback . . . . .	68
BangBang . . . . .	32
MotionController . . . . .	118
PID . . . . .	155
PIDFF . . . . .	162
TakeBackHalf . . . . .	197
FeedForward . . . . .	70
FeedForward::ff_config_t . . . . .	72
Filter . . . . .	73
ExponentialMovingAverage . . . . .	66
MovingAverage . . . . .	124
Flywheel . . . . .	74
GenericAuto . . . . .	91
GraphDrawer . . . . .	93
PurePursuit::hermite_point . . . . .	95
screen::LabelConfig . . . . .	100
Lift< T > . . . . .	101
Lift< T >::lift_cfg_t . . . . .	105
Logger . . . . .	106
MotionController::m_profile_cfg_t . . . . .	110
Mat2 . . . . .	111
MecanumDrive . . . . .	112
MecanumDrive::mecanumdrive_config_t . . . . .	116
motion_t . . . . .	117
Odometry3Wheel::odometry3wheel_cfg_t . . . . .	131
OdometryBase . . . . .	132
Odometry3Wheel . . . . .	127
OdometryTank . . . . .	141
screen::Page . . . . .	149
AutoChooser . . . . .	24
FlywheelPage . . . . .	79
MotionControllerPage . . . . .	123
screen::FunctionPage . . . . .	89
screen::OdometryPage . . . . .	139
screen::PIDPage . . . . .	166
screen::StatsPage . . . . .	194
screen::WidgetPage . . . . .	235
parallel_runner_info . . . . .	153
PurePursuit::Path . . . . .	154
PID::pid_config_t . . . . .	160
point_t . . . . .	168
pose_t . . . . .	170
Rect . . . . .	174
robot_specs_t . . . . .	178
screen::ScreenData . . . . .	180
screen::ScreenRect . . . . .	181
Serializer . . . . .	182
screen::SizedWidget . . . . .	186
SliderCfg . . . . .	187
screen::SliderConfig . . . . .	188
screen::SliderWidget . . . . .	188
PurePursuit::spline . . . . .	192
TankDrive . . . . .	200
screen::TextConfig . . . . .	214
trapezoid_profile_segment_t . . . . .	215

TrapezoidProfile . . . . .	216
Vector2D . . . . .	225
screen::WidgetConfig . . . . .	233



# Chapter 4

## Class Index

### 4.1 Class List

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

AndCondition . . . . .	21
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 . . . . .	22
AutoChooser . . . . .	24
AutoCommand . . . . .	29
BangBang . . . . .	32
BasicSolenoidSet . . . . .	34
BasicSpinCommand . . . . .	37
BasicStopCommand . . . . .	39
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 . . . . .	42
screen::ButtonConfig . . . . .	44
screen::ButtonWidget	
Widget that does something when you tap it. The function is only called once when you first tap it	44
screen::CheckboxConfig . . . . .	46
CommandController . . . . .	47
Condition . . . . .	51
CustomEncoder . . . . .	52
DelayCommand . . . . .	55
DriveForwardCommand . . . . .	57
DriveStopCommand . . . . .	60
DriveToPointCommand . . . . .	62
AutoChooser::entry_t . . . . .	64
ExponentialMovingAverage . . . . .	66
Feedback . . . . .	68
FeedForward . . . . .	70
FeedForward::ff_config_t . . . . .	72
Filter . . . . .	73
Flywheel . . . . .	74
FlywheelPage . . . . .	79
FlywheelStopCommand . . . . .	81
FlywheelStopMotorsCommand . . . . .	83

FlywheelStopNonTasksCommand	85
FunctionCommand	86
FunctionCondition	
FunctionCondition	is a quick and dirty <a href="#">Condition</a> to wrap some expression that should be evaluated at runtime
	88
screen::FunctionPage	
Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything	89
GenericAuto	91
GraphDrawer	93
PurePursuit::hermite_point	95
IfTimePassed	
IfTimePassed	tests based on time since the command controller was constructed. Returns true if elapsed time > time_s
	96
InOrder	
InOrder	runs its commands sequentially then continues. How to handle timeout in this case.
Automatically set it to sum of commands timeouts?	98
screen::LabelConfig	100
Lift< T >	101
Lift< T >::lift_cfg_t	105
Logger	
Class to simplify writing to files	106
MotionController::m_profile_cfg_t	110
Mat2	111
MecanumDrive	112
MecanumDrive::mecanumdrive_config_t	116
motion_t	117
MotionController	118
MotionControllerPage	123
MovingAverage	124
Odometry3Wheel	127
Odometry3Wheel::odometry3wheel_cfg_t	131
OdometryBase	132
screen::OdometryPage	
Page	that shows odometry position and rotation and a map (if an sd card with the file is on)
	139
OdometryTank	141
OdomSetPosition	144
OrCondition	147
screen::Page	
Page	describes one part of the screen slideshow
	149
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
	150
parallel_runner_info	153
PurePursuit::Path	154
PID	155
PID::pid_config_t	160
PIDFF	162
screen::PIDPage	
PIDPage	provides a way to tune a pid controller on the screen
	166
point_t	168
pose_t	170
PurePursuitCommand	172
Rect	174
RepeatUntil	176
robot_specs_t	178
screen::ScreenData	
Holds the data that will be passed to the screen thread you probably shouldnt have to use it	180

screen::ScreenRect . . . . .	181
Serializer	
Serializes Arbitrary data to a file on the SD Card . . . . .	182
screen::SizedWidget . . . . .	186
SliderCfg . . . . .	187
screen::SliderConfig . . . . .	188
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 . . . . .	188
SpinRPMCommand . . . . .	190
PurePursuit::spline . . . . .	192
screen::StatsPage	
Draws motor stats and battery stats to the screen . . . . .	194
TakeBackHalf	
A velocity controller . . . . .	197
TankDrive . . . . .	200
screen::TextConfig . . . . .	214
TimesTestedCondition . . . . .	214
trapezoid_profile_segment_t . . . . .	215
TrapezoidProfile . . . . .	216
TurnDegreesCommand . . . . .	220
TurnToHeadingCommand . . . . .	223
Vector2D . . . . .	225
WaitUntilCondition	
Waits until the condition is true . . . . .	229
WaitUntilUpToSpeedCommand . . . . .	231
screen::WidgetConfig . . . . .	233
screen::WidgetPage . . . . .	235



# Chapter 5

## File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

include/robot_specs.h . . . . .	239
include/subsystems/custom_encoder.h . . . . .	240
include/subsystems/flywheel.h . . . . .	241
include/subsystems/layout.h . . . . .	242
include/subsystems/lift.h . . . . .	243
include/subsystems/mecanum_drive.h . . . . .	246
include/subsystems/screen.h . . . . .	251
include/subsystems/tank_drive.h . . . . .	256
include/subsystems/odometry/odometry_3wheel.h . . . . .	248
include/subsystems/odometry/odometry_base.h . . . . .	249
include/subsystems/odometry/odometry_tank.h . . . . .	250
include/utils/auto_chooser.h . . . . .	258
include/utils/generic_auto.h . . . . .	279
include/utils/geometry.h . . . . .	281
include/utils/graph_drawer.h . . . . .	283
include/utils/logger.h . . . . .	284
include/utils/math_util.h . . . . .	285
include/utils/moving_average.h . . . . .	289
include/utils/pure_pursuit.h . . . . .	290
include/utils/serializer.h . . . . .	292
include/utils/vector2d.h . . . . .	294
include/utils/command_structure/auto_command.h . . . . .	259
include/utils/command_structure/basic_command.h . . . . .	262
include/utils/command_structure/command_controller.h . . . . .	263
include/utils/command_structure/delay_command.h . . . . .	265
include/utils/command_structure/drive_commands.h . . . . .	265
include/utils/command_structure/flywheel_commands.h . . . . .	268
include/utils/controls/bang_bang.h . . . . .	270
include/utils/controls/feedback_base.h . . . . .	270
include/utils/controls/feedforward.h . . . . .	271
include/utils/controls/motion_controller.h . . . . .	273
include/utils/controls/pid.h . . . . .	275
include/utils/controls/pidff.h . . . . .	276
include/utils/controls/take_back_half.h . . . . .	277
include/utils/controls/trapezoid_profile.h . . . . .	278

src/subsystems/ <a href="#">custom_encoder.cpp</a>	297
src/subsystems/ <a href="#">flywheel.cpp</a>	297
src/subsystems/ <a href="#">mecanum_drive.cpp</a>	298
src/subsystems/ <a href="#">screen.cpp</a>	299
src/subsystems/ <a href="#">tank_drive.cpp</a>	300
src/subsystems/odometry/ <a href="#">odometry_3wheel.cpp</a>	298
src/subsystems/odometry/ <a href="#">odometry_base.cpp</a>	299
src/subsystems/odometry/ <a href="#">odometry_tank.cpp</a>	299
src/utils/ <a href="#">auto_chooser.cpp</a>	301
src/utils/ <a href="#">generic_auto.cpp</a>	307
src/utils/ <a href="#">graph_drawer.cpp</a>	308
src/utils/ <a href="#">logger.cpp</a>	308
src/utils/ <a href="#">math_util.cpp</a>	309
src/utils/ <a href="#">moving_average.cpp</a>	312
src/utils/ <a href="#">pure_pursuit.cpp</a>	312
src/utils/ <a href="#">serializer.cpp</a>	313
src/utils/ <a href="#">trapezoid_profile.cpp</a>	314
src/utils/ <a href="#">vector2d.cpp</a>	316
src/utils/command_structure/ <a href="#">auto_command.cpp</a>	302
src/utils/command_structure/ <a href="#">basic_command.cpp</a>	302
src/utils/command_structure/ <a href="#">command_controller.cpp</a>	303
src/utils/command_structure/ <a href="#">drive_commands.cpp</a>	303
src/utils/command_structure/ <a href="#">flywheel_commands.cpp</a>	303
src/utils/controls/ <a href="#">bang_bang.cpp</a>	304
src/utils/controls/ <a href="#">feedforward.cpp</a>	304
src/utils/controls/ <a href="#">motion_controller.cpp</a>	305
src/utils/controls/ <a href="#">pid.cpp</a>	306
src/utils/controls/ <a href="#">pidff.cpp</a>	306
src/utils/controls/ <a href="#">take_back_half.cpp</a>	307

# Chapter 6

## Namespace Documentation

### 6.1 PurePursuit Namespace Reference

#### Classes

- struct `hermite_point`
- class `Path`
- struct `spline`

#### Functions

- `std::vector< point_t > line_circle_intersections (point_t center, double r, point_t point1, point_t point2)`
- `point_t get_lookahead (const std::vector< point_t > &path, pose_t robot_loc, double radius)`
- `std::vector< point_t > inject_path (const std::vector< point_t > &path, double spacing)`
- `std::vector< point_t > smooth_path (const std::vector< point_t > &path, double weight_data, double weight_smooth, double tolerance)`
- `std::vector< point_t > smooth_path_cubic (const std::vector< point_t > &path, double res)`
- `std::vector< point_t > smooth_path_hermite (const std::vector< hermite_point > &path, double step)`
- `double estimate_remaining_dist (const std::vector< point_t > &path, pose_t robot_pose, double radius)`

#### 6.1.1 Function Documentation

##### 6.1.1.1 `estimate_remaining_dist()`

```
double PurePursuit::estimate_remaining_dist (
    const std::vector< point_t > & path,
    pose_t robot_pose,
    double radius ) [extern]
```

Estimates the remaining distance from the robot's position to the end, by "searching" for the robot along the path and running a "connect the dots" distance algorithm

#### Parameters

<code>path</code>	The pure pursuit path the robot is following
<code>robot_pose</code>	The robot's current position
<code>radius</code>	Pure pursuit "radius", used to search for the robot along the path

**Returns**

A rough estimate of the remaining distance

**6.1.1.2 get\_lookahead()**

```
point_t PurePursuit::get_lookahead (
    const std::vector< point_t > & path,
    pose_t robot_loc,
    double radius ) [extern]
```

Selects a look ahead from all the intersections in the path.

**6.1.1.3 inject\_path()**

```
std::vector< point_t > PurePursuit::inject_path (
    const std::vector< point_t > & path,
    double spacing ) [extern]
```

Injects points in a path without changing the curvature with a certain spacing.

**6.1.1.4 line\_circle\_intersections()**

```
std::vector< point_t > PurePursuit::line_circle_intersections (
    point_t center,
    double r,
    point_t point1,
    point_t point2 ) [extern]
```

Returns points of the intersections of a line segment and a circle. The line segment is defined by two points, and the circle is defined by a center and radius.

**6.1.1.5 smooth\_path()**

```
std::vector< point_t > PurePursuit::smooth_path (
    const std::vector< point_t > & path,
    double weight_data,
    double weight_smooth,
    double tolerance ) [extern]
```

Returns a smoothed path maintaining the start and end of the path.

Weight data is how much weight to update the data (alpha) Weight smooth is how much weight to smooth the coordinates (beta) Tolerance is how much change per iteration is necessary to continue iterating.

Honestly have no idea if/how this works. <https://medium.com/@jaems33/understanding-robot-motion-path-planning-101-113a2a2f3a>

### 6.1.1.6 smooth\_path\_cubic()

```
std::vector< point_t > PurePursuit::smooth_path_cubic (
    const std::vector< point_t > & path,
    double res ) [extern]
```

### 6.1.1.7 smooth\_path\_hermite()

```
std::vector< point_t > PurePursuit::smooth_path_hermite (
    const std::vector< hermite_point > & path,
    double steps ) [extern]
```

Interpolates a smooth path given a list of waypoints using hermite splines. For more information: <https://www.youtube.com/watch?v=hG0p4XgePSA>.

#### Parameters

<i>path</i>	The path of hermite points to interpolate.
<i>steps</i>	The number of points interpolated between points.

#### Returns

The smoothed path.

## 6.2 screen Namespace Reference

### Classes

- struct [ButtonConfig](#)
- class [ButtonWidget](#)

*Widget that does something when you tap it. The function is only called once when you first tap it.*
- struct [CheckboxConfig](#)
- class [FunctionPage](#)

*Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything.*
- struct [LabelConfig](#)
- class [OdometryPage](#)

*a page that shows odometry position and rotation and a map (if an sd card with the file is on)*
- class [Page](#)

*Page describes one part of the screen slideshow.*
- class [PIDPage](#)

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

*The ScreenData class holds the data that will be passed to the screen thread you probably shouldnt have to use it.*
- struct [ScreenRect](#)
- struct [SizedWidget](#)
- struct [SliderConfig](#)
- class [SliderWidget](#)

*Widget that updates a double value. Updates by reference so watch out for race conditions cuz the screen stuff lives on another thread.*

- class [StatsPage](#)

*Draws motor stats and battery stats to the screen.*
- struct [TextConfig](#)
- struct [WidgetConfig](#)
- class [WidgetPage](#)

## TypeDefs

- using [update\\_func\\_t](#) = std::function<void(bool, int, int)>

*type of function needed for update*
- using [draw\\_func\\_t](#) = std::function<void(vex::brain::lcd &screen, bool, unsigned int)>

*type of function needed for draw*

## Functions

- void [draw\\_widget](#) ([WidgetConfig](#) &widget, [ScreenRect](#) rect)
- void [start\\_screen](#) (vex::brain::lcd &screen, std::vector< [Page](#) \* > pages, int first\_page=0)

*Start the screen background task. Once you start this, no need to draw to the screen manually elsewhere.*
- void [next\\_page](#) ()
- void [prev\\_page](#) ()
- void [stop\\_screen](#) ()

*stops the screen. If you have a drive team that hates fun call this at the start of opcontrol*
- void [draw\\_label](#) (vex::brain::lcd &scr, std::string lbl, [ScreenRect](#) rect)
- void [draw\\_widget](#) (vex::brain::lcd &scr, [WidgetConfig](#) &widget, [ScreenRect](#) rect)
- int [in\\_to\\_px](#) (double in)

### 6.2.1 Typedef Documentation

#### 6.2.1.1 [draw\\_func\\_t](#)

```
using screen::draw\_func\_t = std::function<void(vex::brain::lcd &screen, bool, unsigned int)>
type of function needed for draw
```

#### 6.2.1.2 [update\\_func\\_t](#)

```
using screen::update\_func\_t = std::function<void(bool, int, int)>
type of function needed for update
```

### 6.2.2 Function Documentation

#### 6.2.2.1 [draw\\_label\(\)](#)

```
void screen::draw_label (
    vex::brain::lcd & scr,
    std::string lbl,
    ScreenRect rect )
```

**6.2.2.2 draw\_widget() [1/2]**

```
void screen::draw_widget (
    vex::brain::lcd & scr,
    WidgetConfig & widget,
    ScreenRect rect )
```

**6.2.2.3 draw\_widget() [2/2]**

```
void screen::draw_widget (
    WidgetConfig & widget,
    ScreenRect rect )
```

**6.2.2.4 in\_to\_px()**

```
int screen::in_to_px (
    double in )
```

**6.2.2.5 next\_page()**

```
void screen::next_page ( )
```

**6.2.2.6 prev\_page()**

```
void screen::prev_page ( )
```

**6.2.2.7 start\_screen()**

```
void screen::start_screen (
    vex::brain::lcd & screen,
    std::vector< Page * > pages,
    int first_page = 0 )
```

Start the screen background task. Once you start this, no need to draw to the screen manually elsewhere.

`start_screen` begins a screen. only call this once per program (a good place is `vexcodeInit`) This is a set and forget type function. You don't have to wait on it or start it in a new thread

**Parameters**

<code>screen</code>	reference to the vex screen
<code>pages</code>	drawing pages
<code>first_page</code>	optional, which page to start the program at. by default 0
<code>screen</code>	the brain screen
<code>pages</code>	the list of pages in your UI slideshow
<code>first_page</code>	the page to start on (by default 0)

### 6.2.2.8 `stop_screen()`

```
void screen::stop_screen ( )
```

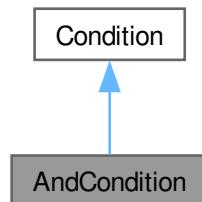
stops the screen. If you have a drive team that hates fun call this at the start of opcontrol

# Chapter 7

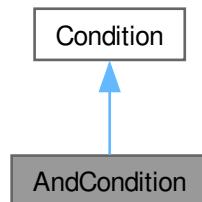
## Class Documentation

### 7.1 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 \(Condition \\*b\)](#)

### 7.1.1 Constructor & Destructor Documentation

#### 7.1.1.1 [AndCondition\(\)](#)

```
AndCondition::AndCondition (
    Condition * A,
    Condition * B ) [inline]
```

### 7.1.2 Member Function Documentation

#### 7.1.2.1 [test\(\)](#)

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

Implements [Condition](#).

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

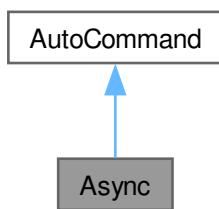
- [src/utils/command\\_structure/auto\\_command.cpp](#)

## 7.2 Async Class Reference

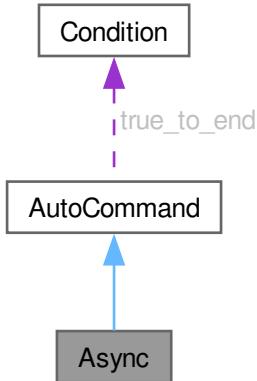
[Async](#) runs a command asynchronously will simply let it go and never look back THIS HAS A VERY NICHE USE CASE. THINK ABOUT IF YOU REALLY NEED IT.

```
#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`

## 7.2.1 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.

## 7.2.2 Constructor & Destructor Documentation

### 7.2.2.1 Async()

```
Async::Async (   
    AutoCommand * cmd ) [inline]
```

## 7.2.3 Member Function Documentation

### 7.2.3.1 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](#).

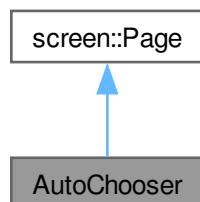
The documentation for this class was generated from the following files:

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

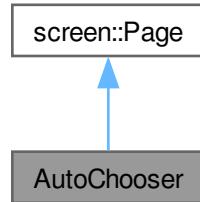
## 7.3 AutoChooser Class Reference

```
#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`

### 7.3.1 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.

## 7.3.2 Constructor & Destructor Documentation

### 7.3.2.1 AutoChooser()

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

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

#### Parameters

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

## 7.3.3 Member Function Documentation

### 7.3.3.1 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](#).

### 7.3.3.2 get\_choice()

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

Get the currently selected auto choice

#### Returns

the identifier to the auto path

Return the selected autonomous

### 7.3.3.3 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**

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

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

## 7.3.4 Member Data Documentation

### 7.3.4.1 choice

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

the current choice of auto

### 7.3.4.2 height

```
const size_t AutoChooser::height = 220 [static], [protected]
```

### 7.3.4.3 list

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

< a list of all possible auto choices

### 7.3.4.4 width

```
const size_t AutoChooser::width = 380 [static], [protected]
```

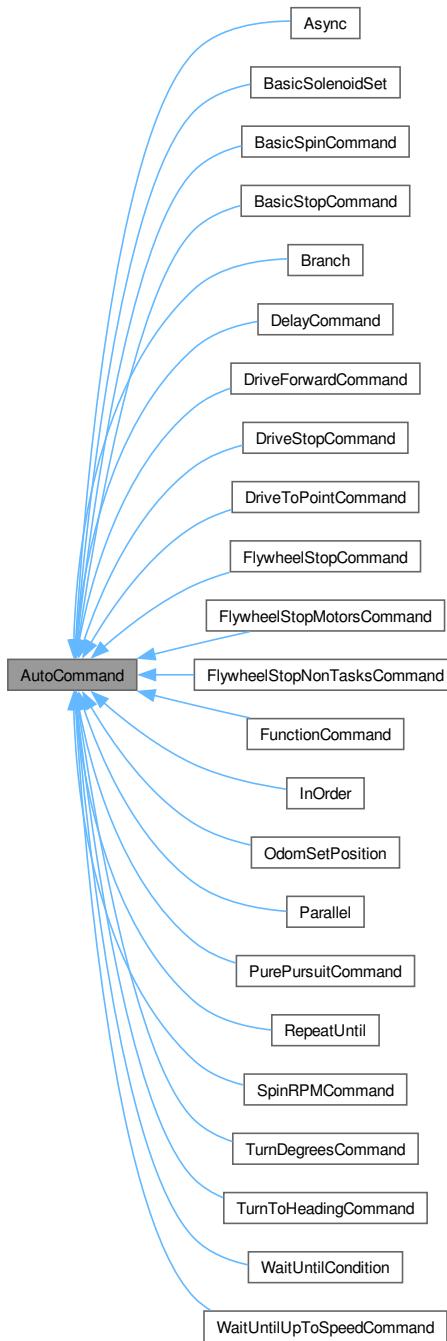
The documentation for this class was generated from the following files:

- [include/utils/auto\\_chooser.h](#)
- [src/utils/auto\\_chooser.cpp](#)

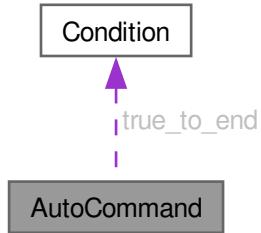
## 7.4 AutoCommand Class Reference

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

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`

### 7.4.1 Member Function Documentation

#### 7.4.1.1 `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`.

#### 7.4.1.2 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](#)

#### 7.4.1.3 withCancelCondition()

```
AutoCommand * AutoCommand::withCancelCondition ( Condition * true_to_end ) [inline]
```

#### 7.4.1.4 withTimeout()

```
AutoCommand * AutoCommand::withTimeout ( double t_seconds ) [inline]
```

## 7.4.2 Member Data Documentation

### 7.4.2.1 default\_timeout

```
constexpr double AutoCommand::default_timeout = 10.0 [static], [constexpr]
```

### 7.4.2.2 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...

### 7.4.2.3 true\_to\_end

```
Condition* AutoCommand::true_to_end = nullptr
```

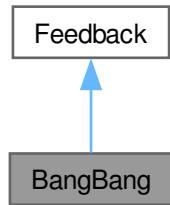
The documentation for this class was generated from the following file:

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

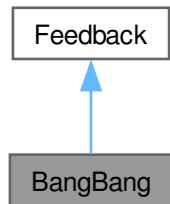
## 7.5 BangBang Class Reference

```
#include <bang_bang.h>
```

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

## 7.5.1 Constructor & Destructor Documentation

### 7.5.1.1 BangBang()

```
BangBang::BangBang (
    double threshold,
    double low,
    double high )
```

## 7.5.2 Member Function Documentation

### 7.5.2.1 get()

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

#### Returns

the last saved result from the feedback controller

Implements [Feedback](#).

### 7.5.2.2 init()

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

Initialize the feedback controller for a movement

#### Parameters

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

Implements [Feedback](#).

### 7.5.2.3 is\_on\_target()

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

#### Returns

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

#### 7.5.2.4 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](#).

#### 7.5.2.5 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](#).

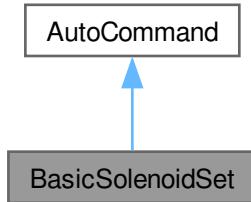
The documentation for this class was generated from the following files:

- include/utils/controls/[bang\\_bang.h](#)
- src/utils/controls/[bang\\_bang.cpp](#)

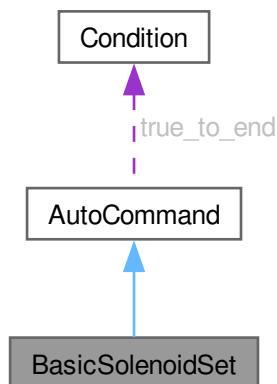
## 7.6 BasicSolenoidSet Class Reference

```
#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

## 7.6.1 Detailed Description

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

## 7.6.2 Constructor & Destructor Documentation

### 7.6.2.1 [BasicSolenoidSet\(\)](#)

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

Construct a new [BasicSolenoidSet](#) Command.

#### Parameters

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

## 7.6.3 Member Function Documentation

### 7.6.3.1 [run\(\)](#)

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

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

#### Returns

True Command runs once

Reimplemented from [AutoCommand](#).

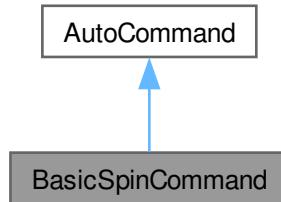
The documentation for this class was generated from the following files:

- include/utils/command\_structure/[basic\\_command.h](#)
- src/utils/command\_structure/[basic\\_command.cpp](#)

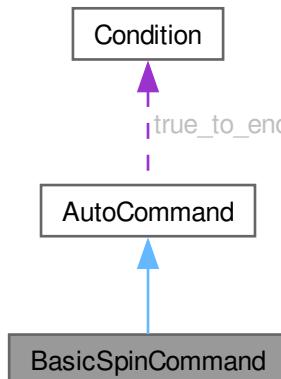
## 7.7 BasicSpinCommand Class Reference

```
#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](#)

### 7.7.1 Detailed Description

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

### 7.7.2 Member Enumeration Documentation

#### 7.7.2.1 type

enum [BasicSpinCommand::type](#)

Enumerator

percent	
voltage	
veocity	

### 7.7.3 Constructor & Destructor Documentation

#### 7.7.3.1 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

**7.7.4 Member Function Documentation****7.7.4.1 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](#).

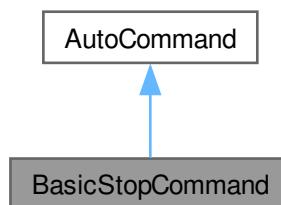
The documentation for this class was generated from the following files:

- include/utils/command\_structure/[basic\\_command.h](#)
- src/utils/command\_structure/[basic\\_command.cpp](#)

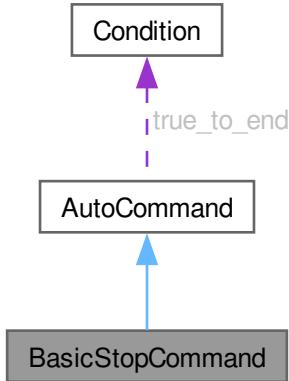
**7.8 BasicStopCommand Class Reference**

```
#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

### 7.8.1 Detailed Description

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

## 7.8.2 Constructor & Destructor Documentation

### 7.8.2.1 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

## 7.8.3 Member Function Documentation

### 7.8.3.1 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:

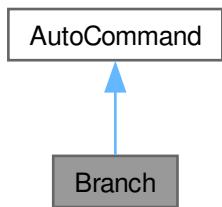
- include/utils/command\_structure/[basic\\_command.h](#)
- src/utils/command\_structure/[basic\\_command.cpp](#)

## 7.9 Branch Class Reference

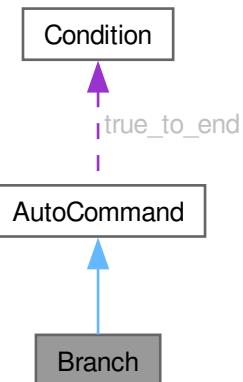
[Branch](#) chooses from multiple options at runtime. the function decider returns an index into the choices vector If you wish to make no choice and skip this section, return NO\_CHOICE; any choice that is out of bounds set to NO\_CHOICE.

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

Inheritance diagram for Branch:



Collaboration diagram for Branch:



### Public Member Functions

- `Branch (Condition *cond, AutoCommand *false_choice, AutoCommand *true_choice)`
- `~Branch ()`
- `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`

## 7.9.1 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.

## 7.9.2 Constructor & Destructor Documentation

### 7.9.2.1 [Branch\(\)](#)

```
Branch::Branch (
    Condition * cond,
    AutoCommand * false_choice,
    AutoCommand * true_choice )
```

### 7.9.2.2 [~Branch\(\)](#)

```
Branch::~Branch ( )
```

## 7.9.3 Member Function Documentation

### 7.9.3.1 [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](#).

### 7.9.3.2 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](#)

## 7.10 screen::ButtonConfig Struct Reference

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

#### Public Attributes

- std::function< void()> [onclick](#)

### 7.10.1 Member Data Documentation

#### 7.10.1.1 onclick

```
std::function<void()> screen::ButtonConfig::onclick
```

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

- include/subsystems/[screen.h](#)

## 7.11 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*

### 7.11.1 Detailed Description

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

### 7.11.2 Constructor & Destructor Documentation

#### 7.11.2.1 `ButtonWidget()` [1/2]

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

Create a Button widget.

##### Parameters

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

#### 7.11.2.2 `ButtonWidget()` [2/2]

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

Create a Button widget.

##### Parameters

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

### 7.11.3 Member Function Documentation

#### 7.11.3.1 draw()

```
void screen::ButtonWidget::draw (
    vex::brain::lcd & scr,
    bool first_draw,
    unsigned int frame_number )
```

draws the button to the screen

#### 7.11.3.2 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](#)

## 7.12 screen::CheckboxConfig Struct Reference

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

#### Public Attributes

- std::function< void(bool)> [onupdate](#)

## 7.12.1 Member Data Documentation

### 7.12.1.1 onupdate

```
std::function<void(bool)> screen::CheckboxConfig::onupdate
```

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

- include/subsystems/screen.h

## 7.13 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 \(\)](#)

### 7.13.1 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.

## 7.13.2 Constructor & Destructor Documentation

### 7.13.2.1 CommandController() [1/2]

```
CommandController::CommandController ( ) [inline]
```

Create an empty [CommandController](#). Add Command with [CommandController::add\(\)](#)

### 7.13.2.2 CommandController() [2/2]

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

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

**Parameters**

<code>cmds</code>	<input type="checkbox"/>
-------------------	--------------------------

**7.13.3 Member Function Documentation****7.13.3.1 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**

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

**7.13.3.2 add() [2/3]**

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

Adds a command to the queue

**Parameters**

<code>cmd</code>	the <code>AutoCommand</code> we want to add to our list
<code>timeout_seconds</code>	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 $\leq 0$ no time out will be applied

Add multiple commands to the queue. No timeout here.

**Parameters**

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

**7.13.3.3 add() [3/3]**

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

Add multiple commands to the queue. No timeout here.

**Parameters**

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

Add multiple commands to the queue. No timeout here.

**Parameters**

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

**7.13.3.4 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
-----------------------	--

**7.13.3.5 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
-----------	--

**7.13.3.6 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

### 7.13.3.7 run()

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

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

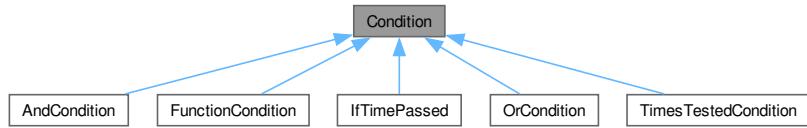
The documentation for this class was generated from the following files:

- include/utils/command\_structure/[command\\_controller.h](#)
- src/utils/command\_structure/[command\\_controller.cpp](#)

## 7.14 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](#)

### 7.14.1 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

### 7.14.2 Member Function Documentation

#### 7.14.2.1 And()

```
Condition * Condition::And (
    Condition * b )
```

### 7.14.2.2 Or()

```
Condition * Condition::Or (  
    Condition * b )
```

### 7.14.2.3 test()

```
virtual bool Condition::test () [pure virtual]
```

Implemented in [TimesTestedCondition](#), [FunctionCondition](#), [IfTimePassed](#), [OrCondition](#), and [AndCondition](#).

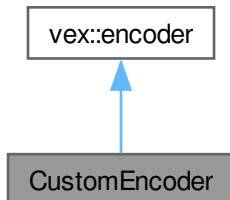
The documentation for this class was generated from the following files:

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

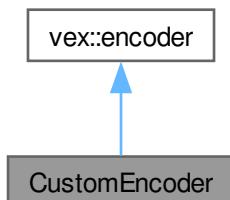
## 7.15 CustomEncoder Class Reference

```
#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)`

### 7.15.1 Detailed Description

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

### 7.15.2 Constructor & Destructor Documentation

#### 7.15.2.1 CustomEncoder()

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

Construct an encoder with a custom number of ticks

##### Parameters

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

### 7.15.3 Member Function Documentation

#### 7.15.3.1 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

### 7.15.3.2 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

### 7.15.3.3 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

### 7.15.3.4 setRotation()

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

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

#### Parameters

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

### 7.15.3.5 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

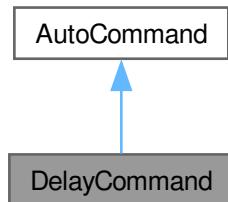
The documentation for this class was generated from the following files:

- include/subsystems/[custom\\_encoder.h](#)
- src/subsystems/[custom\\_encoder.cpp](#)

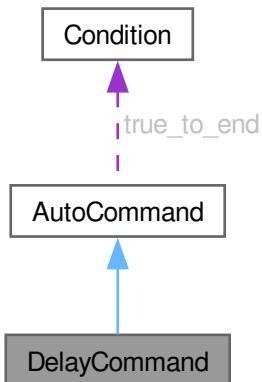
## 7.16 DelayCommand Class Reference

```
#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`

## 7.16.1 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

## 7.16.2 Constructor & Destructor Documentation

### 7.16.2.1 DelayCommand()

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

Construct a delay command

#### Parameters

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

## 7.16.3 Member Function Documentation

### 7.16.3.1 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](#).

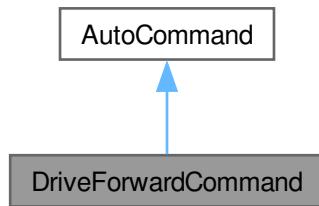
The documentation for this class was generated from the following file:

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

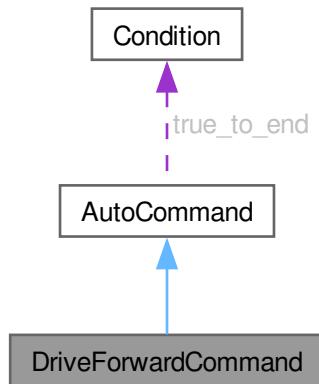
## 7.17 DriveForwardCommand Class Reference

```
#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`

## 7.17.1 Detailed Description

`AutoCommand` wrapper class for the `drive_forward` function in the `TankDrive` class

## 7.17.2 Constructor & Destructor Documentation

### 7.17.2.1 `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

## 7.17.3 Member Function Documentation

### 7.17.3.1 `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](#).

### 7.17.3.2 `run()`

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

Run drive\_forward Overrides run from [AutoCommand](#)

**Returns**

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

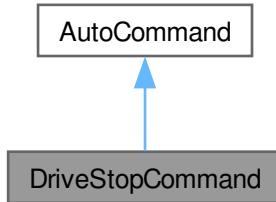
The documentation for this class was generated from the following files:

- include/utils/command\_structure/[drive\\_commands.h](#)
- src/utils/command\_structure/[drive\\_commands.cpp](#)

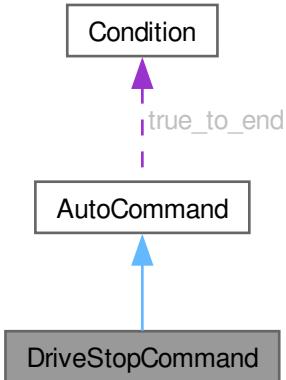
## 7.18 DriveStopCommand Class Reference

```
#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

## 7.18.1 Detailed Description

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

## 7.18.2 Constructor & Destructor Documentation

### 7.18.2.1 `DriveStopCommand()`

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

Construct a DriveStop Command

#### Parameters

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

## 7.18.3 Member Function Documentation

### 7.18.3.1 `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](#).

### 7.18.3.2 `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](#).

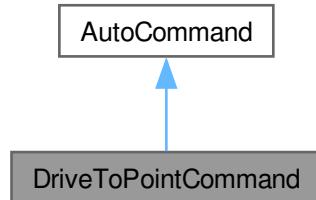
The documentation for this class was generated from the following files:

- include/utils/command\_structure/[drive\\_commands.h](#)
- src/utils/command\_structure/[drive\\_commands.cpp](#)

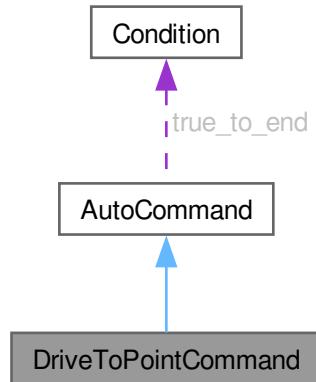
## 7.19 DriveToPointCommand Class Reference

```
#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`

## 7.19.1 Detailed Description

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

## 7.19.2 Constructor & Destructor Documentation

### 7.19.2.1 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

<code>drive_sys</code>	the drive system we are commanding
<code>feedback</code>	the feedback controller we are using to execute the drive
<code>x</code>	where to drive in the x dimension
<code>y</code>	where to drive in the y dimension
<small>Generated by Doxygen</small>	The direction to drive
<code>max_speed</code>	0 -> 1 percentage of the drive systems speed to drive at

### 7.19.2.2 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

## 7.19.3 Member Function Documentation

### 7.19.3.1 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](#).

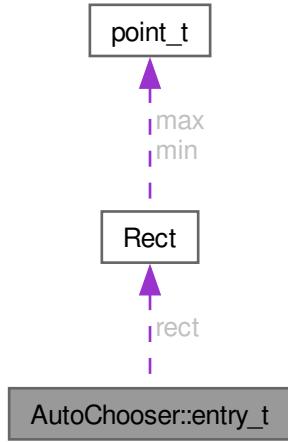
The documentation for this class was generated from the following files:

- include/utils/command\_structure/[drive\\_commands.h](#)
- src/utils/command\_structure/[drive\\_commands.cpp](#)

## 7.20 AutoChooser::entry\_t Struct Reference

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

Collaboration diagram for AutoChooser::entry\_t:



## Public Attributes

- `Rect rect`
- `std::string name`

### 7.20.1 Detailed Description

`entry_t` is a datatype used to store information that the chooser knows about an auto selection button

### 7.20.2 Member Data Documentation

#### 7.20.2.1 name

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

name of the auto represented by the block

#### 7.20.2.2 rect

```
Rect AutoChooser::entry_t::rect
```

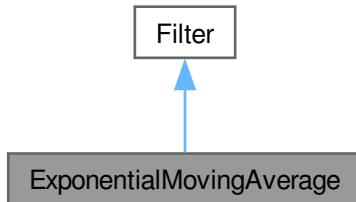
The documentation for this struct was generated from the following file:

- `include/utils/auto_chooser.h`

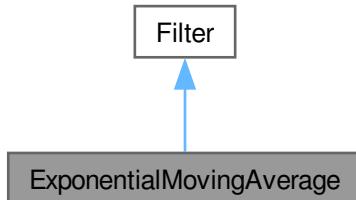
## 7.21 ExponentialMovingAverage Class Reference

```
#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 \(\)](#)

### 7.21.1 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.

## 7.21.2 Constructor & Destructor Documentation

### 7.21.2.1 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
--------------------	---

### 7.21.2.2 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

## 7.21.3 Member Function Documentation

### 7.21.3.1 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](#).

### 7.21.3.2 get\_size()

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

How many samples the average is made from

**Returns**

the number of samples used to calculate this average

**7.21.3.3 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](#).

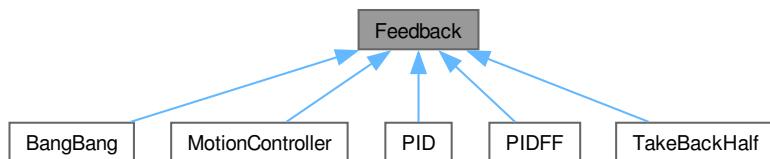
The documentation for this class was generated from the following files:

- include/utils/[moving\\_average.h](#)
- src/utils/[moving\\_average.cpp](#)

## 7.22 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

### 7.22.1 Detailed Description

Interface so that subsystems can easily switch between feedback loops

#### Author

Ryan McGee

#### Date

9/25/2022

### 7.22.2 Member Function Documentation

#### 7.22.2.1 get()

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

##### Returns

the last saved result from the feedback controller

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

#### 7.22.2.2 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](#).

#### 7.22.2.3 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](#).

**7.22.2.4 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](#).

**7.22.2.5 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](#)

**7.23 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.*

### 7.23.1 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

### 7.23.2 Constructor & Destructor Documentation

#### 7.23.2.1 FeedForward()

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

Creates a `FeedForward` object.

#### Parameters

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

### 7.23.3 Member Function Documentation

#### 7.23.3.1 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](#)

## 7.24 FeedForward::ff\_config\_t Struct Reference

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

### Public Attributes

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

#### 7.24.1 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

- $kV * \text{desired velocity}$
- $kA * \text{desired acceleration}$
- $kG$

## 7.24.2 Member Data Documentation

### 7.24.2.1 kA

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

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

### 7.24.2.2 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.

### 7.24.2.3 kS

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

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

### 7.24.2.4 kV

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

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

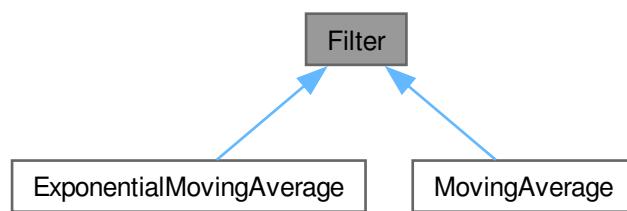
The documentation for this struct was generated from the following file:

- include/utils/controls/[feedforward.h](#)

## 7.25 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

### 7.25.1 Detailed Description

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

### 7.25.2 Member Function Documentation

#### 7.25.2.1 `add_entry()`

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

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

#### 7.25.2.2 `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](#)

## 7.26 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)

**7.26.1 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

**7.26.2 Constructor & Destructor Documentation****7.26.2.1 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

**7.26.3 Member Function Documentation****7.26.3.1 get\_motors()**

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

Returns the motors

**Returns**

the motors used to run the flywheel

### 7.26.3.2 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

### 7.26.3.3 getRPM()

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

return the velocity of the flywheel

### 7.26.3.4 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

### 7.26.3.5 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()});`

### 7.26.3.6 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

**7.26.3.7 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
------------------	-----------------------

**7.26.3.8 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

### 7.26.3.9 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

### 7.26.3.10 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

## 7.26.4 Friends And Related Symbol Documentation

### 7.26.4.1 FlywheelPage

```
friend class FlywheelPage [friend]
```

### 7.26.4.2 spinRPMTTask

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

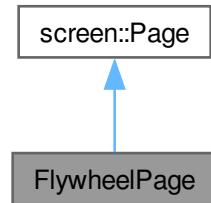
Runs a thread that keeps track of updating flywheel RPM and controlling it accordingly

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

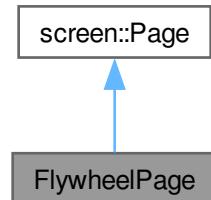
- include/subsystems/flywheel.h
- src/subsystems/flywheel.cpp

## 7.27 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 &scren, bool, unsigned int) override

### Static Public Attributes

- static const size\_t [window\\_size](#) = 40

### 7.27.1 Constructor & Destructor Documentation

#### 7.27.1.1 FlywheelPage()

```
FlywheelPage::FlywheelPage (
```

```
    const Flywheel & fw ) [inline]
```

## 7.27.2 Member Function Documentation

### 7.27.2.1 draw()

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

#### See also

[Page::draw](#)

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

### 7.27.2.2 update()

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

#### See also

[Page::update](#)

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

## 7.27.3 Member Data Documentation

### 7.27.3.1 window\_size

```
const size_t FlywheelPage::window_size = 40 [static]
```

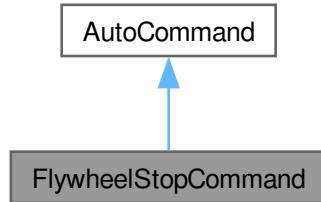
The documentation for this class was generated from the following file:

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

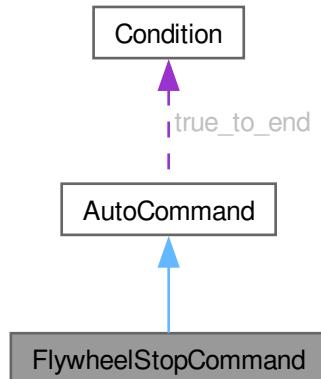
## 7.28 FlywheelStopCommand Class Reference

```
#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

## 7.28.1 Detailed Description

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

## 7.28.2 Constructor & Destructor Documentation

### 7.28.2.1 FlywheelStopCommand()

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

Construct a [FlywheelStopCommand](#)

#### Parameters

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

## 7.28.3 Member Function Documentation

### 7.28.3.1 run()

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

Run stop Overrides run from [AutoCommand](#)

#### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

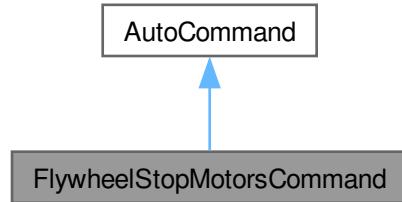
The documentation for this class was generated from the following files:

- include/utils/command\_structure/[flywheel\\_commands.h](#)
- src/utils/command\_structure/[flywheel\\_commands.cpp](#)

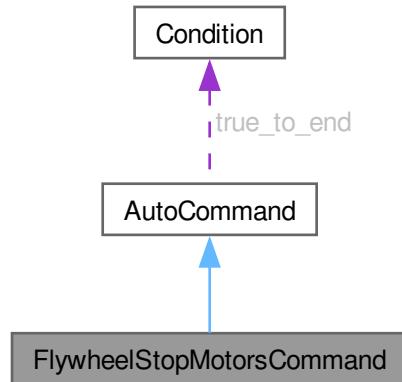
## 7.29 FlywheelStopMotorsCommand Class Reference

```
#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

## 7.29.1 Detailed Description

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

## 7.29.2 Constructor & Destructor Documentation

### 7.29.2.1 FlywheelStopMotorsCommand()

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

Construct a FlywheelStopMotors Command

#### Parameters

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

## 7.29.3 Member Function Documentation

### 7.29.3.1 run()

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

Run stop Overrides run from [AutoCommand](#)

#### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

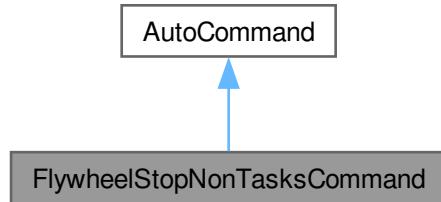
The documentation for this class was generated from the following files:

- include/utils/command\_structure/[flywheel\\_commands.h](#)
- src/utils/command\_structure/[flywheel\\_commands.cpp](#)

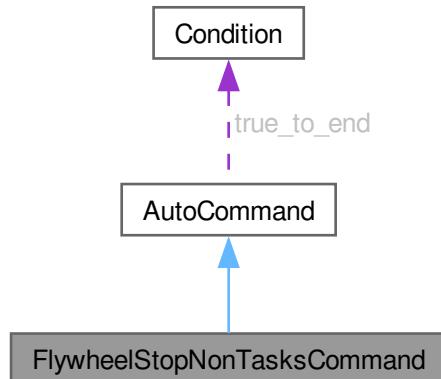
## 7.30 FlywheelStopNonTasksCommand Class Reference

```
#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

#### 7.30.1 Detailed Description

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

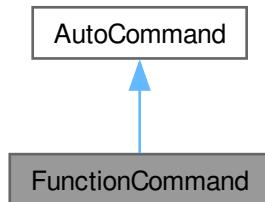
The documentation for this class was generated from the following files:

- include/utils/command\_structure/[flywheel\\_commands.h](#)
- src/utils/command\_structure/[flywheel\\_commands.cpp](#)

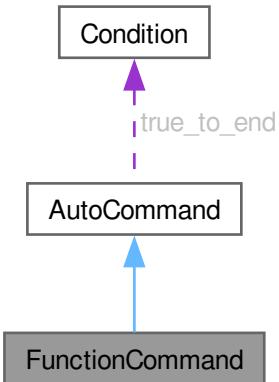
## 7.31 FunctionCommand Class Reference

```
#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](#)

## 7.31.1 Detailed Description

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

## 7.31.2 Constructor & Destructor Documentation

### 7.31.2.1 [FunctionCommand\(\)](#)

```
FunctionCommand::FunctionCommand (
    std::function< bool(void)> f ) [inline]
```

## 7.31.3 Member Function Documentation

### 7.31.3.1 [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:

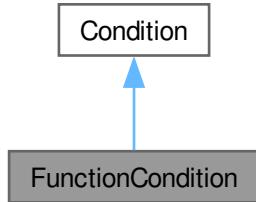
- [include/utils/command\\_structure/auto\\_command.h](#)

## 7.32 FunctionCondition Class Reference

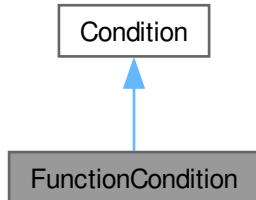
[FunctionCondition](#) is a quick and dirty [Condition](#) to wrap some expression that should be evaluated at runtime.

```
#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\)](#)

#### 7.32.1 Detailed Description

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

## 7.32.2 Constructor & Destructor Documentation

### 7.32.2.1 FunctionCondition()

```
FunctionCondition::FunctionCondition (
    std::function< bool()> cond,
    std::function< void(void)> timeout = []() {} ) [inline]
```

## 7.32.3 Member Function Documentation

### 7.32.3.1 test()

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

Implements [Condition](#).

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

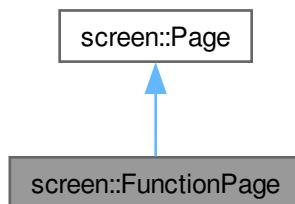
- include/utils/command\_structure/[auto\\_command.h](#)
- src/utils/command\_structure/[auto\\_command.cpp](#)

## 7.33 screen::FunctionPage Class Reference

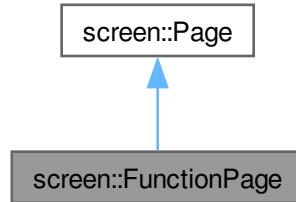
Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything.

```
#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*

### 7.33.1 Detailed Description

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

### 7.33.2 Constructor & Destructor Documentation

#### 7.33.2.1 FunctionPage()

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

Creates a function page.

[FunctionPage](#).

#### Parameters

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

### 7.33.3 Member Function Documentation

#### 7.33.3.1 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](#).

#### 7.33.3.2 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](#)

## 7.34 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 \(int ms\)](#)

### 7.34.1 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

### 7.34.2 Member Function Documentation

#### 7.34.2.1 add()

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

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

Parameters

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

#### 7.34.2.2 add\_async()

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

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

Parameters

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

#### 7.34.2.3 add\_delay()

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

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

Parameters

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

#### 7.34.2.4 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

## 7.35 GraphDrawer Class Reference

```
#include <graph_drawer.h>
```

#### 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)

### 7.35.1 Constructor & Destructor Documentation

#### 7.35.1.1 GraphDrawer()

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

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

#### Parameters

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

## 7.35.2 Member Function Documentation

### 7.35.2.1 add\_samples() [1/2]

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

add\_samples adds a point to the graph, removing one from the back

#### Parameters

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

### 7.35.2.2 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
---------------	---

### 7.35.2.3 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

## 7.36 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`

### 7.36.1 Detailed Description

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

### 7.36.2 Member Function Documentation

#### 7.36.2.1 getPoint()

```
point_t PurePursuit::hermite_point::getPoint () const [inline]
```

#### 7.36.2.2 getTangent()

```
Vector2D PurePursuit::hermite_point::getTangent () const [inline]
```

### 7.36.3 Member Data Documentation

#### 7.36.3.1 dir

```
double PurePursuit::hermite_point::dir
```

#### 7.36.3.2 mag

```
double PurePursuit::hermite_point::mag
```

### 7.36.3.3 x

```
double PurePursuit::hermite_point::x
```

### 7.36.3.4 y

```
double PurePursuit::hermite_point::y
```

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

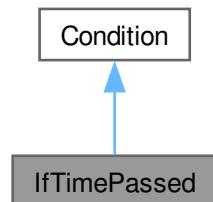
- include/utils/pure\_pursuit.h

## 7.37 IfTimePassed Class Reference

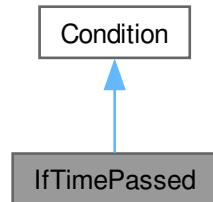
IfTimePassed tests based on time since the command controller was constructed. Returns true if elapsed time > time\_s.

```
#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)

### 7.37.1 Detailed Description

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

### 7.37.2 Constructor & Destructor Documentation

#### 7.37.2.1 [IfTimePassed\(\)](#)

```
IfTimePassed::IfTimePassed (
    double time_s )
```

### 7.37.3 Member Function Documentation

#### 7.37.3.1 [test\(\)](#)

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

Implements [Condition](#).

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

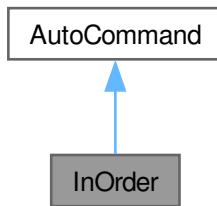
- include/utils/command\_structure/[auto\\_command.h](#)
- src/utils/command\_structure/[auto\\_command.cpp](#)

## 7.38 InOrder Class Reference

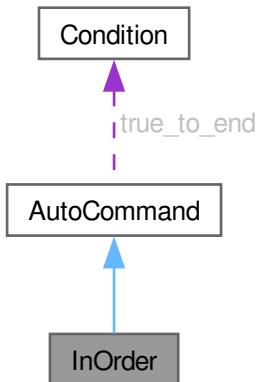
`InOrder` runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timeouts?

```
#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](#)

### 7.38.1 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?

### 7.38.2 Constructor & Destructor Documentation

#### 7.38.2.1 [InOrder\(\) \[1/3\]](#)

```
InOrder::InOrder (
    const InOrder & other ) [default]
```

#### 7.38.2.2 [InOrder\(\) \[2/3\]](#)

```
InOrder::InOrder (
    std::queue< AutoCommand * > cmdss )
```

#### 7.38.2.3 [InOrder\(\) \[3/3\]](#)

```
InOrder::InOrder (
    std::initializer_list< AutoCommand * > cmdss )
```

### 7.38.3 Member Function Documentation

#### 7.38.3.1 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](#).

#### 7.38.3.2 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](#)

## 7.39 screen::LabelConfig Struct Reference

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

### Public Attributes

- `std::string label`

#### 7.39.1 Member Data Documentation

##### 7.39.1.1 label

```
std::string screen::LabelConfig::label
```

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

- [include/subsystems/screen.h](#)

## 7.40 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))

### 7.40.1 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

### 7.40.2 Constructor & Destructor Documentation

#### 7.40.2.1 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**

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

**7.40.3 Member Function Documentation****7.40.3.1 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

**7.40.3.2 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

**7.40.3.3 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).

#### 7.40.3.4 get\_async()

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

#### Returns

whether or not the background thread is running the lift

#### 7.40.3.5 get\_setpoint()

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

#### Returns

The current setpoint for the lift

#### 7.40.3.6 hold()

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

Target the class's setpoint. Calculate the [PID](#) output and set the lift motors accordingly.

#### 7.40.3.7 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.

#### 7.40.3.8 set\_async()

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

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

**Parameters**

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

**7.40.3.9 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

**7.40.3.10 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
---------------	-----------------------------------

**7.40.3.11 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

**7.40.3.12 set\_setpoint()**

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

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

**Parameters**

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

**Returns**

True if the pid has reached the setpoint

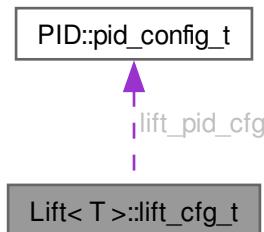
The documentation for this class was generated from the following file:

- include/subsystems/lift.h

## 7.41 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](#)

### 7.41.1 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

## 7.41.2 Member Data Documentation

### 7.41.2.1 down\_speed

```
template<typename T >
double Lift< T >::lift_cfg_t::down_speed
```

### 7.41.2.2 lift\_pid\_cfg

```
template<typename T >
PID::pid_config_t Lift< T >::lift_cfg_t::lift_pid_cfg
```

### 7.41.2.3 softstop\_down

```
template<typename T >
double Lift< T >::lift_cfg_t::softstop_down
```

### 7.41.2.4 softstop\_up

```
template<typename T >
double Lift< T >::lift_cfg_t::softstop_up
```

### 7.41.2.5 up\_speed

```
template<typename T >
double Lift< T >::lift_cfg_t::up_speed
```

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

- include/subsystems/[lift.h](#)

## 7.42 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 Logln (const std::string &s)`  
*Write a string and newline to the log.*
- `void Logln (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*

### 7.42.1 Detailed Description

Class to simplify writing to files.

### 7.42.2 Constructor & Destructor Documentation

#### 7.42.2.1 `Logger()` [1/2]

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

Create a logger that will save to a file.

##### Parameters

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

#### 7.42.2.2 `Logger()` [2/2]

```
Logger::Logger (
    const Logger & l ) [delete]
```

copying not allowed

### 7.42.3 Member Function Documentation

#### 7.42.3.1 Log() [1/2]

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

Write a string to the log.

##### Parameters

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

#### 7.42.3.2 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

#### 7.42.3.3 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

#### 7.42.3.4 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

### 7.42.3.5 `Login()` [1/2]

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

Write a string and newline to the log.

#### Parameters

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

### 7.42.3.6 `Login()` [2/2]

```
void Logger::Login (
    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

### 7.42.3.7 `operator=()`

```
Logger & Logger::operator= (
    const Logger & l ) [delete]
```

copying not allowed

## 7.42.4 Member Data Documentation

### 7.42.4.1 `MAX_FORMAT_LEN`

```
constexpr int Logger::MAX_FORMAT_LEN = 512 [static], [constexpr]
```

maximum size for a string to be before it's written

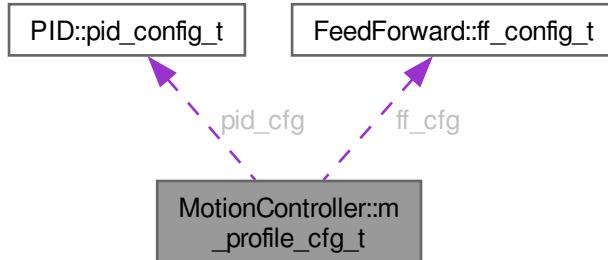
The documentation for this class was generated from the following files:

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

## 7.43 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*

### 7.43.1 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.

## 7.43.2 Member Data Documentation

### 7.43.2.1 accel

```
double MotionController::m_profile_cfg_t::accel
```

the most acceleration the robot can do

### 7.43.2.2 ff\_cfg

```
FeedForward::ff_config_t MotionController::m_profile_cfg_t::ff_cfg
```

configuration parameters for the internal

### 7.43.2.3 max\_v

```
double MotionController::m_profile_cfg_t::max_v
```

the maximum velocity the robot can drive

### 7.43.2.4 pid\_cfg

```
PID::pid_config_t MotionController::m_profile_cfg_t::pid_cfg
```

configuration parameters for the internal PID controller

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

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

## 7.44 Mat2 Struct Reference

```
#include <geometry.h>
```

### 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

**7.44.1 Member Function Documentation****7.44.1.1 FromRotationDegrees()**

```
static Mat2 Mat2::FromRotationDegrees (
    double degrees ) [inline], [static]
```

**7.44.1.2 operator\*()**

```
point_t Mat2::operator* (
    const point_t p ) const [inline]
```

**7.44.2 Member Data Documentation****7.44.2.1 X11**

```
double Mat2::X11
```

**7.44.2.2 X12**

```
double Mat2::X12
```

**7.44.2.3 X21**

```
double Mat2::X21
```

**7.44.2.4 X22**

```
double Mat2::X22
```

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

- include/utils/[geometry.h](#)

**7.45 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)

### 7.45.1 Detailed Description

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

### 7.45.2 Constructor & Destructor Documentation

#### 7.45.2.1 `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

### 7.45.3 Member Function Documentation

#### 7.45.3.1 `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>Generated by Doxygen</i>	
<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.

### 7.45.3.2 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

### 7.45.3.3 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)

### 7.45.3.4 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:

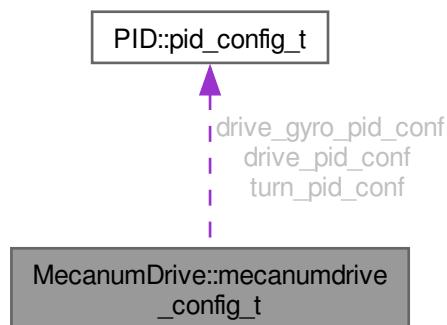
- include/subsystems/mecanum\_drive.h

- src/subsystems/mecanum\_drive.cpp

## 7.46 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`

### 7.46.1 Detailed Description

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

### 7.46.2 Member Data Documentation

#### 7.46.2.1 drive\_gyro\_pid\_conf

`PID::pid_config_t MecanumDrive::mecanumdrive_config_t::drive_gyro_pid_conf`

#### 7.46.2.2 drive\_pid\_conf

`PID::pid_config_t MecanumDrive::mecanumdrive_config_t::drive_pid_conf`

#### 7.46.2.3 drive\_wheel\_diam

```
double MecanumDrive::mecanumdrive_config_t::drive_wheel_diam
```

#### 7.46.2.4 lateral\_wheel\_diam

```
double MecanumDrive::mecanumdrive_config_t::lateral_wheel_diam
```

#### 7.46.2.5 turn\_pid\_conf

```
PID::pid_config_t MecanumDrive::mecanumdrive_config_t::turn_pid_conf
```

#### 7.46.2.6 wheelbase\_width

```
double MecanumDrive::mecanumdrive_config_t::wheelbase_width
```

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

- include/subsystems/mecanum\_drive.h

## 7.47 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*

### 7.47.1 Detailed Description

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

### 7.47.2 Member Data Documentation

#### 7.47.2.1 accel

```
double motion_t::accel
```

1d acceleration at this point in time

### 7.47.2.2 pos

```
double motion_t::pos
```

1d position at this point in time

### 7.47.2.3 vel

```
double motion_t::vel
```

1d velocity at this point in time

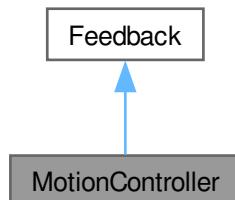
The documentation for this struct was generated from the following file:

- [include/utils/controls/trapezoid\\_profile.h](#)

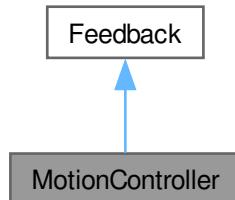
## 7.48 MotionController Class Reference

```
#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](#)

## 7.48.1 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

## 7.48.2 Constructor & Destructor Documentation

### 7.48.2.1 MotionController()

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

Construct a new Motion Controller object.

**Parameters**

<i>config</i>	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
---------------	--

**7.48.3 Member Function Documentation****7.48.3.1 get()**

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

**Returns**

the last saved result from the feedback controller

Implements [Feedback](#).

**7.48.3.2 get\_motion()**

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

**Returns**

The current position, velocity and acceleration setpoints

**7.48.3.3 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](#).

**7.48.3.4 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](#).

**7.48.3.5 Page()**

```
screen::Page * MotionController::Page ( )
```

**7.48.3.6 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 limiet

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](#).

**7.48.3.7 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 it's steady-state velocity at 'pct' speed.

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

#### 7.48.3.8 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](#).

### 7.48.4 Friends And Related Symbol Documentation

#### 7.48.4.1 MotionControllerPage

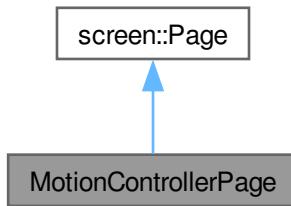
```
friend class MotionControllerPage  [friend]
```

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

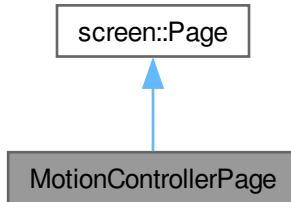
- include/utils/controls/[motion\\_controller.h](#)
- src/utils/controls/[motion\\_controller.cpp](#)

## 7.49 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)*

### 7.49.1 Constructor & Destructor Documentation

#### 7.49.1.1 MotionControllerPage()

```
MotionControllerPage::MotionControllerPage (
    const MotionController & mc ) [inline]
```

## 7.49.2 Member Function Documentation

### 7.49.2.1 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](#).

### 7.49.2.2 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](#).

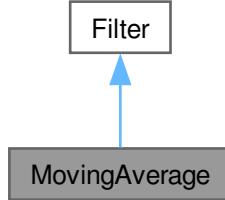
The documentation for this class was generated from the following file:

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

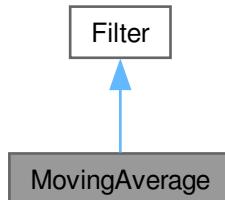
## 7.50 MovingAverage Class Reference

```
#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](#)

### 7.50.1 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).

## 7.50.2 Constructor & Destructor Documentation

### 7.50.2.1 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
--------------------	---

### 7.50.2.2 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

## 7.50.3 Member Function Documentation

### 7.50.3.1 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](#).

### 7.50.3.2 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

**7.50.3.3 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](#).

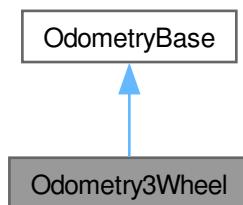
The documentation for this class was generated from the following files:

- include/utils/[moving\\_average.h](#)
- src/utils/[moving\\_average.cpp](#)

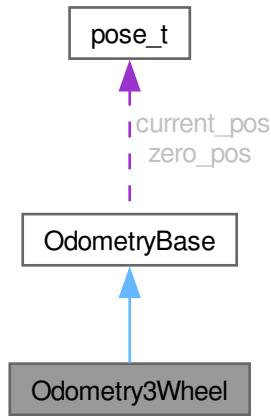
## 7.51 Odometry3Wheel Class Reference

```
#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`

### 7.51.1 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:

```
+Y ----- ^ | | | | | | | O | | | | | | === | | ----- | +-----> + X
```

Where O is the center of rotation. The robot will monitor the changes in rotation of these wheels and calculate the robot's X, Y and rotation on the field.

This is a "set and forget" class, meaning once the object is created, the robot will immediately begin tracking its movement in the background.

#### Author

Ryan McGee

#### Date

Oct 31 2022

### 7.51.2 Constructor & Destructor Documentation

#### 7.51.2.1 [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

**7.51.3 Member Function Documentation****7.51.3.1 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

**7.51.3.2 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](#)

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

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

### Public Attributes

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

### 7.52.1 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

### 7.52.2 Member Data Documentation

#### 7.52.2.1 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

#### 7.52.2.2 wheel\_diam

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

the diameter of the tracking wheel

#### 7.52.2.3 wheelbase\_dist

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

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

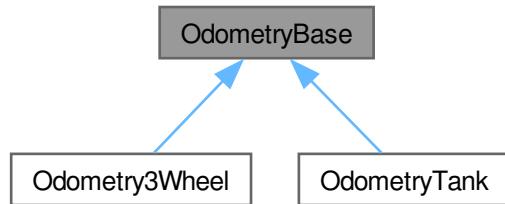
The documentation for this struct was generated from the following file:

- include/subsystems/odometry/[odometry\\_3wheel.h](#)

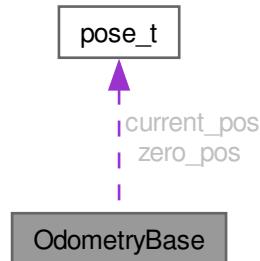
## 7.53 OdometryBase Class Reference

```
#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`

## 7.53.1 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

## 7.53.2 Constructor & Destructor Documentation

### 7.53.2.1 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
-----------------	---

**7.53.3 Member Function Documentation****7.53.3.1 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.

**7.53.3.2 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.

**7.53.3.3 get\_accel()**

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

Get the current acceleration

**Returns**

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

**7.53.3.4 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>)

### 7.53.3.5 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)

### 7.53.3.6 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

### 7.53.3.7 get\_speed()

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

Get the current speed

#### Returns

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

### 7.53.3.8 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

**7.53.3.9 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

**7.53.3.10 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](#).

**7.53.3.11 SetPositionCmd()**

```
AutoCommand * OdometryBase::SetPositionCmd (
    const pose_t & newpos = zero_pos )
```

**7.53.3.12 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.

### 7.53.3.13 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](#).

## 7.53.4 Member Data Documentation

### 7.53.4.1 accel

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

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

### 7.53.4.2 ang\_accel\_deg

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

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

### 7.53.4.3 ang\_speed\_deg

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

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

#### 7.53.4.4 current\_pos

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

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

#### 7.53.4.5 end\_task

```
bool OdometryBase::end_task = false
```

end\_task is true if we instruct the odometry thread to shut down

#### 7.53.4.6 handle

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

handle to the vex task that is running the odometry code

#### 7.53.4.7 mut

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

Mutex to control multithreading

#### 7.53.4.8 speed

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

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

#### 7.53.4.9 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:

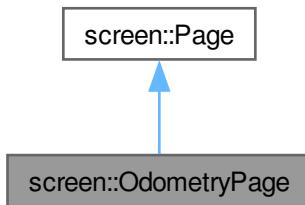
- include/subsystems/odometry/[odometry\\_base.h](#)
- src/subsystems/odometry/[odometry\\_base.cpp](#)

## 7.54 screen::OdometryPage Class Reference

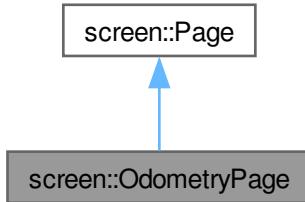
a page that shows odometry position and rotation and a map (if an sd card with the file is on)

```
#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](#)

### 7.54.1 Detailed Description

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

## 7.54.2 Constructor & Destructor Documentation

### 7.54.2.1 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

## 7.54.3 Member Function Documentation

### 7.54.3.1 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](#).

### 7.54.3.2 update()

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

#### See also

[Page::update](#)

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

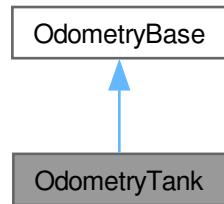
The documentation for this class was generated from the following files:

- include/subsystems/[screen.h](#)
- src/subsystems/[screen.cpp](#)

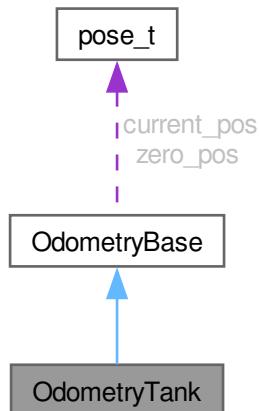
## 7.55 OdometryTank Class Reference

```
#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](#)

## 7.55.1 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.

## 7.55.2 Constructor & Destructor Documentation

### 7.55.2.1 [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> .

**7.55.2.2 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> .

**7.55.2.3 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> .

### 7.55.3 Member Function Documentation

#### 7.55.3.1 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](#).

#### 7.55.3.2 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](#).

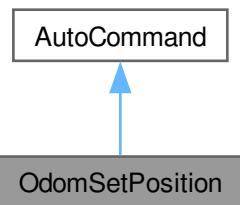
The documentation for this class was generated from the following files:

- include/subsystems/odometry/[odometry\\_tank.h](#)
- src/subsystems/odometry/[odometry\\_tank.cpp](#)

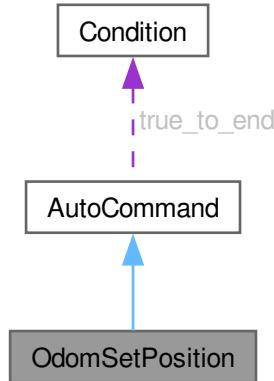
## 7.56 OdomSetPosition Class Reference

```
#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`

### 7.56.1 Detailed Description

[AutoCommand](#) wrapper class for the set\_position function in the Odometry class

## 7.56.2 Constructor & Destructor Documentation

### 7.56.2.1 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

### 7.56.3 Member Function Documentation

#### 7.56.3.1 run()

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

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

**Returns**

true when execution is complete, false otherwise

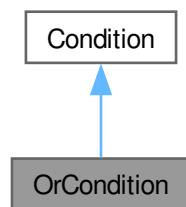
Reimplemented from [AutoCommand](#).

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

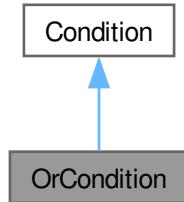
- include/utils/command\_structure/[drive\\_commands.h](#)
- src/utils/command\_structure/[drive\\_commands.cpp](#)

## 7.57 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 \(Condition \\*b\)](#)

## 7.57.1 Constructor & Destructor Documentation

### 7.57.1.1 [OrCondition\(\)](#)

```
OrCondition::OrCondition (
    Condition * A,
    Condition * B ) [inline]
```

## 7.57.2 Member Function Documentation

### 7.57.2.1 [test\(\)](#)

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

Implements [Condition](#).

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

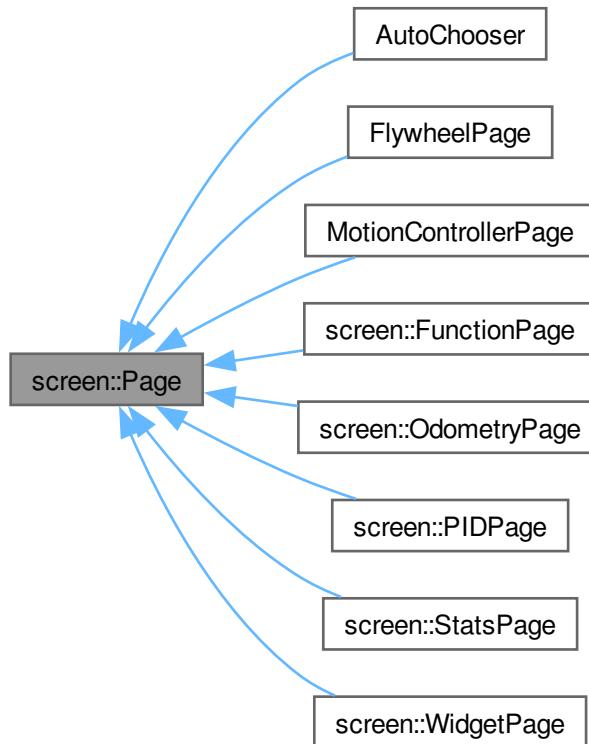
- [src/utils/command\\_structure/auto\\_command.cpp](#)

## 7.58 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)*

### 7.58.1 Detailed Description

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

## 7.58.2 Member Function Documentation

### 7.58.2.1 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](#).

### 7.58.2.2 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:

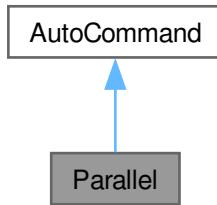
- include/subsystems/[screen.h](#)

## 7.59 Parallel Class Reference

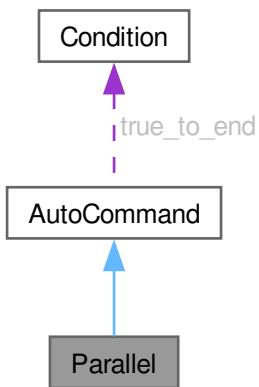
[Parallel](#) runs multiple commands in parallel and waits for all to finish before continuing. if none finish before this command's timeout, it will call `on_timeout` on all children continue.

```
#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

### 7.59.1 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.

### 7.59.2 Constructor & Destructor Documentation

#### 7.59.2.1 [Parallel\(\)](#)

```
Parallel::Parallel ( std::initializer_list< AutoCommand * > cmd )
```

### 7.59.3 Member Function Documentation

#### 7.59.3.1 [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](#).

#### 7.59.3.2 [run\(\)](#)

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

Executes the command Overridden by child classes

#### Returns

true when the command is finished, false otherwise

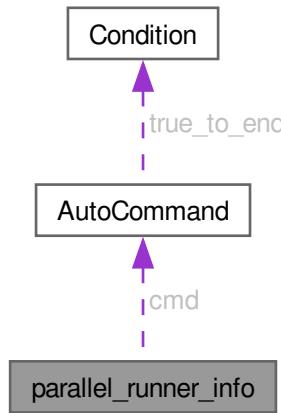
Reimplemented from [AutoCommand](#).

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](#)

## 7.60 parallel\_runner\_info Struct Reference

Collaboration diagram for parallel\_runner\_info:



### Public Attributes

- int `index`
- std::vector< vex::task \* > \* `runners`
- `AutoCommand * cmd`

#### 7.60.1 Member Data Documentation

##### 7.60.1.1 cmd

`AutoCommand* parallel_runner_info::cmd`

##### 7.60.1.2 index

`int parallel_runner_info::index`

##### 7.60.1.3 runners

`std::vector<vex::task *>* parallel_runner_info::runners`

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

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

## 7.61 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 \(\)](#)

### 7.61.1 Detailed Description

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

### 7.61.2 Constructor & Destructor Documentation

#### 7.61.2.1 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

### 7.61.3 Member Function Documentation

#### 7.61.3.1 get\_points()

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

Get the points associated with this [Path](#)

#### 7.61.3.2 get\_radius()

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

Get the radius associated with this [Path](#)

### 7.61.3.3 `is_valid()`

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

Get whether this path will behave as expected

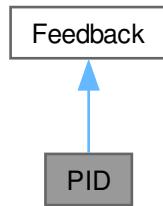
The documentation for this class was generated from the following files:

- [include/utils/pure\\_pursuit.h](#)
- [src/utils/pure\\_pursuit.cpp](#)

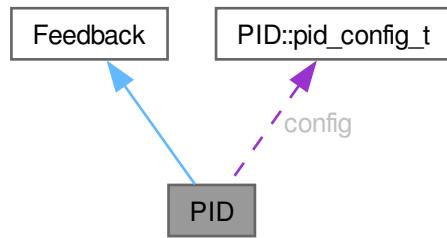
## 7.62 PID Class Reference

```
#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`

### 7.62.1 Detailed Description

#### PID Class

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

`out = kP*error + kI*integral(d Error) + kD*(dError/dt)`

The `PID` object will determine it is "on target" when the error is within the deadband, for a duration of on\_target\_time

#### Author

Ryan McGee

#### Date

4/3/2020

### 7.62.2 Member Enumeration Documentation

#### 7.62.2.1 ERROR\_TYPE

```
enum PID::ERROR_TYPE
```

An enum to distinguish between a linear and angular calculation of `PID` error.

Enumerator

LINEAR	
ANGULAR	

### 7.62.3 Constructor & Destructor Documentation

#### 7.62.3.1 PID()

```
PID::PID (
    pid_config_t & config )
```

Create the [PID](#) object

Parameters

<i>config</i>	the configuration data for this controller
---------------	--

Create the [PID](#) object

### 7.62.4 Member Function Documentation

#### 7.62.4.1 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](#).

#### 7.62.4.2 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

#### 7.62.4.3 `get_sensor_val()`

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

gets the sensor value that we were last updated with

##### Returns

`sensor_val`

#### 7.62.4.4 `get_target()`

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

Get the [PID](#)'s target

##### Returns

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

#### 7.62.4.5 `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](#).

#### 7.62.4.6 `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](#).

**7.62.4.7 reset()**

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

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

**7.62.4.8 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](#).

**7.62.4.9 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

**7.62.4.10 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

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

## 7.62.5 Member Data Documentation

### 7.62.5.1 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](#)

## 7.63 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**

### 7.63.1 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

### 7.63.2 Member Data Documentation

#### 7.63.2.1 d

```
double PID::pid_config_t::d  
derivative coefficient d * derivative(error)
```

#### 7.63.2.2 deadband

```
double PID::pid_config_t::deadband  
at what threshold are we close enough to be finished
```

#### 7.63.2.3 error\_method

```
ERROR_TYPE PID::pid_config_t::error_method  
Linear or angular. whether to do error as a simple subtraction or to wrap
```

#### 7.63.2.4 i

```
double PID::pid_config_t::i  
integral coefficient i * integral(error)
```

#### 7.63.2.5 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
```

### 7.63.2.6 p

```
double PID::pid_config_t::p
```

```
proportional coeffecient p * error()
```

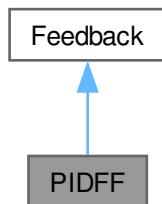
The documentation for this struct was generated from the following file:

- include/utils/controls/pid.h

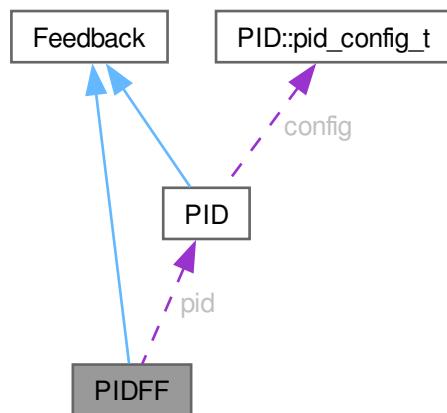
## 7.64 PIDFF Class Reference

```
#include <pidff.h>
```

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`

**7.64.1 Constructor & Destructor Documentation****7.64.1.1 PIDFF()**

```
PIDFF::PIDFF (
    PID::pid_config_t & pid_cfg,
    FeedForward::ff_config_t & ff_cfg )
```

**7.64.2 Member Function Documentation****7.64.2.1 get()**

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

**Returns**

the last saved result from the feedback controller

Implements [Feedback](#).

**7.64.2.2 get\_sensor\_val()**

```
double PIDFF::get_sensor_val () const
```

**7.64.2.3 get\_target()**

```
double PIDFF::get_target () const
```

**7.64.2.4 init()**

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

Initialize the feedback controller for a movement

**Parameters**

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

Initialize the feedback controller for a movement

**Parameters**

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

Implements [Feedback](#).

**7.64.2.5 is\_on\_target()**

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

**Returns**

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

**7.64.2.6 reset()**

```
void PIDFF::reset ( )
```

**7.64.2.7 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](#).

**7.64.2.8 set\_target()**

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

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

**Parameters**

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

**7.64.2.9 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](#).

**7.64.2.10 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

### 7.64.3 Member Data Documentation

#### 7.64.3.1 pid

PID PIDFF::pid

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

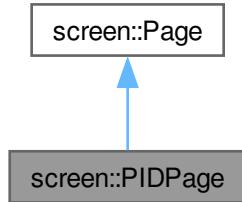
- include/utils/controls/pidff.h
- src/utils/controls/pidff.cpp

## 7.65 screen::PIDPage Class Reference

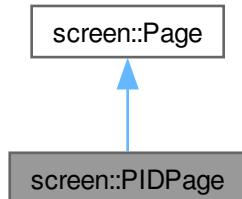
[PIDPage](#) provides a way to tune a pid controller on the screen.

```
#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`

### 7.65.1 Detailed Description

`PIDPage` provides a way to tune a pid controller on the screen.

### 7.65.2 Constructor & Destructor Documentation

#### 7.65.2.1 PIDPage() [1/2]

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

Create a `PIDPage`.

##### Parameters

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

#### 7.65.2.2 PIDPage() [2/2]

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

### 7.65.3 Member Function Documentation

#### 7.65.3.1 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](#).

### 7.65.3.2 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](#)

## 7.66 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*

### 7.66.1 Detailed Description

Data structure representing an X,Y coordinate

## 7.66.2 Member Function Documentation

### 7.66.2.1 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**

<i>other</i>	the point to measure the distance from
--------------	--

**Returns**

the euclidian distance between this and other

**7.66.2.2 operator\*()**

```
point_t point_t::operator* (
    double s ) const [inline]
```

**7.66.2.3 operator+() [1/2]**

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

**7.66.2.4 operator+() [2/2]**

```
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)

**7.66.2.5 operator-() [1/2]**

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

**7.66.2.6 operator-() [2/2]**

```
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)

**7.66.2.7 operator/()**

```
point_t point_t::operator/ (
    double s ) const [inline]
```

**7.66.2.8 operator==( )**

```
bool point_t::operator== (
    const point_t & rhs ) [inline]
```

**7.66.3 Member Data Documentation****7.66.3.1 x**

```
double point_t::x
```

the x position in space

**7.66.3.2 y**

```
double point_t::y
```

the y position in space

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

- [include/utils/geometry.h](#)

**7.67 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*

### 7.67.1 Detailed Description

Describes a single position and rotation

### 7.67.2 Member Function Documentation

#### 7.67.2.1 get\_point()

```
point_t pose_t::get_point ( ) [inline]
```

### 7.67.3 Member Data Documentation

#### 7.67.3.1 rot

```
double pose_t::rot
```

rotation in the world

#### 7.67.3.2 x

```
double pose_t::x
```

x position in the world

#### 7.67.3.3 y

```
double pose_t::y
```

y position in the world

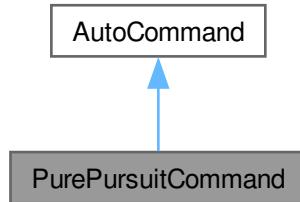
The documentation for this struct was generated from the following file:

- include/utils/[geometry.h](#)

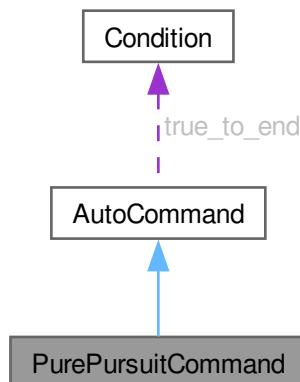
## 7.68 PurePursuitCommand Class Reference

```
#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

## 7.68.1 Detailed Description

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

## 7.68.2 Constructor & Destructor Documentation

### 7.68.2.1 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)

## 7.68.3 Member Function Documentation

### 7.68.3.1 on\_timeout()

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

Reset the drive system when it times out

Reimplemented from [AutoCommand](#).

### 7.68.3.2 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](#)

## 7.69 Rect Struct Reference

```
#include <geometry.h>
```

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](#)

## 7.69.1 Member Function Documentation

### 7.69.1.1 center()

```
point_t Rect::center () const [inline]
```

### 7.69.1.2 contains()

```
bool Rect::contains (
    point_t p ) const [inline]
```

### 7.69.1.3 dimensions()

```
point_t Rect::dimensions () const [inline]
```

### 7.69.1.4 from\_min\_and\_size()

```
static Rect Rect::from_min_and_size (
    point_t min,
    point_t size ) [inline], [static]
```

### 7.69.1.5 height()

```
double Rect::height () const [inline]
```

### 7.69.1.6 width()

```
double Rect::width () const [inline]
```

## 7.69.2 Member Data Documentation

### 7.69.2.1 max

```
point_t Rect::max
```

### 7.69.2.2 min

```
point_t Rect::min
```

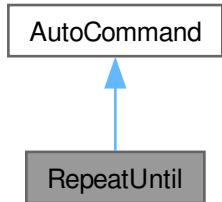
The documentation for this struct was generated from the following file:

- include/utils/[geometry.h](#)

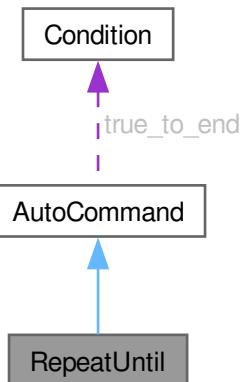
## 7.70 RepeatUntil Class Reference

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

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

## 7.70.1 Constructor & Destructor Documentation

### 7.70.1.1 RepeatUntil() [1/2]

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

RepeatUntil that runs a fixed number of times.

#### Parameters

<i>cmds</i>	the cmds to repeat
<i>repeats</i>	the number of repeats to do

### 7.70.1.2 RepeatUntil() [2/2]

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

RepeatUntil the condition.

#### Parameters

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

## 7.70.2 Member Function Documentation

### 7.70.2.1 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](#).

### 7.70.2.2 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](#).

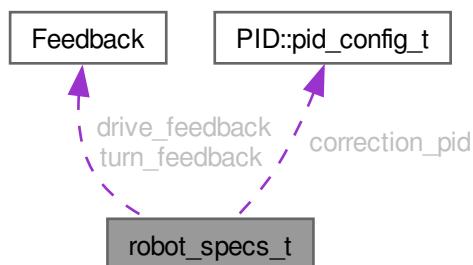
The documentation for this class was generated from the following files:

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

## 7.71 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*

### 7.71.1 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.

### 7.71.2 Member Data Documentation

#### 7.71.2.1 `correction_pid`

`PID::pid_config_t robot_specs_t::correction_pid`

the pid controller to keep the robot driving in as straight a line as possible

#### 7.71.2.2 `dist_between_wheels`

`double robot_specs_t::dist_between_wheels`

the distance between centers of the central drive wheels

#### 7.71.2.3 `drive_correction_cutoff`

`double robot_specs_t::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

#### 7.71.2.4 `drive_feedback`

`Feedback*` `robot_specs_t::drive_feedback`

the default feedback for autonomous driving

#### 7.71.2.5 `odom_gear_ratio`

`double robot_specs_t::odom_gear_ratio`

the ratio of the odometry wheel to the encoder reading odometry data

#### 7.71.2.6 `odom_wheel_diam`

`double robot_specs_t::odom_wheel_diam`

the diameter of the wheels used for

#### 7.71.2.7 `robot_radius`

`double robot_specs_t::robot_radius`

if you were to draw a circle with this radius, the robot would be entirely contained within it

#### 7.71.2.8 `turn_feedback`

`Feedback*` `robot_specs_t::turn_feedback`

the defualt feedback for autonomous turning

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

- `include/robot_specs.h`

## 7.72 `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`

### 7.72.1 Detailed Description

The [ScreenData](#) class holds the data that will be passed to the screen thread you probably shouldnt have to use it.

### 7.72.2 Constructor & Destructor Documentation

#### 7.72.2.1 ScreenData()

```
screen::ScreenData::ScreenData (
    const std::vector< Page * > & m_pages,
    int m_page,
    vex::brain::lcd & m_screen ) [inline]
```

### 7.72.3 Member Data Documentation

#### 7.72.3.1 page

```
int screen::ScreenData::page = 0
```

#### 7.72.3.2 pages

```
std::vector<Page *> screen::ScreenData::pages
```

#### 7.72.3.3 screen

```
vex::brain::lcd screen::ScreenData::screen
```

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

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

## 7.73 screen::ScreenRect Struct Reference

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

### Public Attributes

- `uint32_t x1`
- `uint32_t y1`
- `uint32_t x2`
- `uint32_t y2`

## 7.73.1 Member Data Documentation

### 7.73.1.1 x1

```
uint32_t screen::ScreenRect::x1
```

### 7.73.1.2 x2

```
uint32_t screen::ScreenRect::x2
```

### 7.73.1.3 y1

```
uint32_t screen::ScreenRect::y1
```

### 7.73.1.4 y2

```
uint32_t screen::ScreenRect::y2
```

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

- include/subsystems/[screen.h](#)

## 7.74 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 doesn't 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*

### 7.74.1 Detailed Description

Serializes Arbitrary data to a file on the SD Card.

### 7.74.2 Constructor & Destructor Documentation

#### 7.74.2.1 ~Serializer()

```
Serializer::~Serializer ( ) [inline]
```

Save and close upon destruction (bc of vex, this doesn't always get called when the program ends. To be sure, call `save_to_disk`)

#### 7.74.2.2 Serializer()

```
Serializer::Serializer (
    const std::string & filename,
    bool flush_always = true ) [inline], [explicit]
```

create a [Serializer](#)

##### Parameters

<code>filename</code>	the file to read from. If filename does not exist we will create that file
<code>flush_always</code>	If true, after every write flush to a file. If false, you are responsible for calling <code>save_to_disk</code>

### 7.74.3 Member Function Documentation

#### 7.74.3.1 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

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

##### Returns

the value if found or otherwise

### 7.74.3.2 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

### 7.74.3.3 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

### 7.74.3.4 save\_to\_disk()

void Serializer::save\_to\_disk ( ) const

saves current [Serializer](#) state to disk

forms data bytes then saves to filename this was opened with

### 7.74.3.5 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

**7.74.3.6 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

**7.74.3.7 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

**7.74.3.8 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>s</i>	value of string

### 7.74.3.9 `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

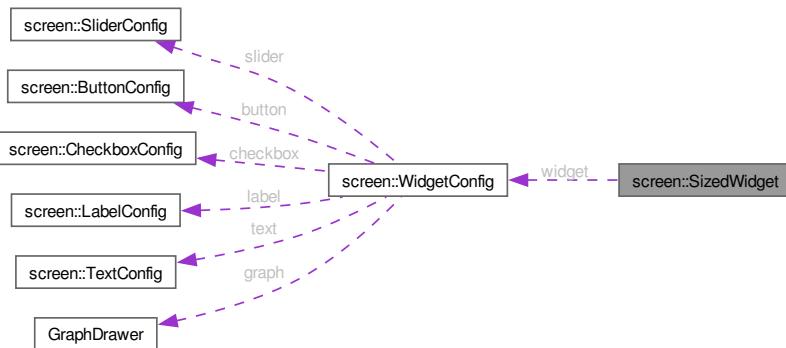
The documentation for this class was generated from the following files:

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

## 7.75 screen::SizedWidget Struct Reference

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

Collaboration diagram for screen::SizedWidget:



#### Public Attributes

- `int size`
- `WidgetConfig & widget`

### 7.75.1 Member Data Documentation

#### 7.75.1.1 size

```
int screen::SizedWidget::size
```

#### 7.75.1.2 widget

```
WidgetConfig& screen::SizedWidget::widget
```

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

- include/subsystems/[screen.h](#)

## 7.76 SliderCfg Struct Reference

```
#include <layout.h>
```

### Public Attributes

- double & [val](#)
- double [min](#)
- double [max](#)

### 7.76.1 Member Data Documentation

#### 7.76.1.1 max

```
double SliderCfg::max
```

#### 7.76.1.2 min

```
double SliderCfg::min
```

#### 7.76.1.3 val

```
double& SliderCfg::val
```

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

- include/subsystems/[layout.h](#)

## 7.77 screen::SliderConfig Struct Reference

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

### Public Attributes

- double & [val](#)
- double [low](#)
- double [high](#)

### 7.77.1 Member Data Documentation

#### 7.77.1.1 [high](#)

```
double screen::SliderConfig::high
```

#### 7.77.1.2 [low](#)

```
double screen::SliderConfig::low
```

#### 7.77.1.3 [val](#)

```
double& screen::SliderConfig::val
```

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

- include/subsystems/[screen.h](#)

## 7.78 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*

### 7.78.1 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.

### 7.78.2 Constructor & Destructor Documentation

#### 7.78.2.1 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

### 7.78.3 Member Function Documentation

#### 7.78.3.1 draw()

```
void screen::SliderWidget::draw (
    vex::brain::lcd & scr,
    bool first_draw,
    unsigned int frame_number )
```

Page::draws the slide to the screen

#### 7.78.3.2 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

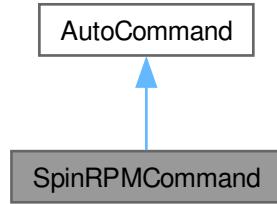
The documentation for this class was generated from the following files:

- include/subsystems/[screen.h](#)
- src/subsystems/[screen.cpp](#)

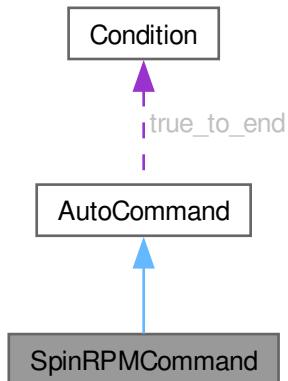
## 7.79 SpinRPMCommand Class Reference

```
#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`

## 7.79.1 Detailed Description

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

## 7.79.2 Constructor & Destructor Documentation

### 7.79.2.1 SpinRPMCommand()

```
SpinRPMCommand::SpinRPMCommand (
    Flywheel & flywheel,
    int rpm )
```

Construct a SpinRPM Command

#### Parameters

<code>flywheel</code>	the flywheel sys to command
<code>rpm</code>	the rpm that we should spin at

File: [flywheel\\_commands.cpp](#) Desc: [insert meaningful desc]

### 7.79.3 Member Function Documentation

#### 7.79.3.1 run()

```
bool SpinRPCommand::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](#)

## 7.80 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](#)

#### 7.80.1 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.

#### 7.80.2 Member Function Documentation

##### 7.80.2.1 [getY\(\)](#)

```
double PurePursuit::spline::getY (
    double x ) [inline]
```

### 7.80.3 Member Data Documentation

#### 7.80.3.1 a

```
double PurePursuit::spline::a
```

#### 7.80.3.2 b

```
double PurePursuit::spline::b
```

#### 7.80.3.3 c

```
double PurePursuit::spline::c
```

#### 7.80.3.4 d

```
double PurePursuit::spline::d
```

#### 7.80.3.5 x\_end

```
double PurePursuit::spline::x_end
```

#### 7.80.3.6 x\_start

```
double PurePursuit::spline::x_start
```

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

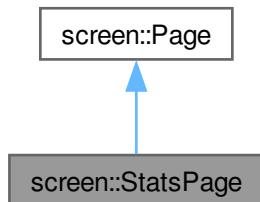
- include/utils/pure\_pursuit.h

## 7.81 screen::StatsPage Class Reference

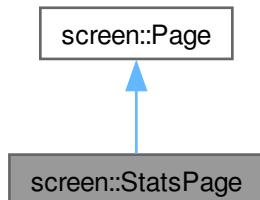
Draws motor stats and battery stats to the screen.

```
#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

### 7.81.1 Detailed Description

Draws motor stats and battery stats to the screen.

## 7.81.2 Constructor & Destructor Documentation

### 7.81.2.1 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
---------------	--

### 7.81.3 Member Function Documentation

#### 7.81.3.1 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](#).

#### 7.81.3.2 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:

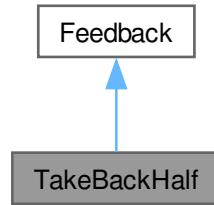
- include/subsystems/[screen.h](#)
- src/subsystems/[screen.cpp](#)

## 7.82 TakeBackHalf Class Reference

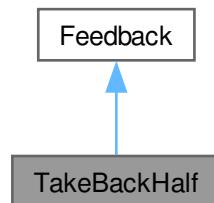
A velocity controller.

```
#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`

### 7.82.1 Detailed Description

A velocity controller.

#### Warning

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

### 7.82.2 Constructor & Destructor Documentation

#### 7.82.2.1 TakeBackHalf()

```
TakeBackHalf::TakeBackHalf (
    double TBH_gain,
    double first_cross_split,
    double on_target_threshold )
```

### 7.82.3 Member Function Documentation

#### 7.82.3.1 get()

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

##### Returns

the last saved result from the feedback controller

Implements [Feedback](#).

#### 7.82.3.2 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](#).

### 7.82.3.3 `is_on_target()`

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

#### Returns

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

### 7.82.3.4 `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](#).

### 7.82.3.5 `update()`

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

Iterate the feedback loop once with an updated sensor value

#### Parameters

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

#### Returns

feedback loop result

Implements [Feedback](#).

## 7.82.4 Member Data Documentation

### 7.82.4.1 `first_cross_split`

```
double TakeBackHalf::first_cross_split
```

#### 7.82.4.2 TBH\_gain

double TakeBackHalf::TBH\_gain

tuned parameter

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

## 7.83 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)

### 7.83.1 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

### 7.83.2 Member Enumeration Documentation

#### 7.83.2.1 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

### 7.83.3 Constructor & Destructor Documentation

#### 7.83.3.1 TankDrive()

```
TankDrive::TankDrive (
    motor_group & left_motors,
    motor_group & right_motors,
    robot_specs_t & config,
    OdometryBase * odom = NULL )
```

Create the `TankDrive` object

##### Parameters

<code>left_motors</code>	left side drive motors
--------------------------	------------------------

**Parameters**

<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

**7.83.4 Member Function Documentation****7.83.4.1 drive\_arena()**

```
void TankDrive::drive_arena (
    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

**7.83.4.2 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

### 7.83.4.3 `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

#### 7.83.4.4 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

#### 7.83.4.5 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)

#### 7.83.4.6 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

**7.83.4.7 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

**7.83.4.8 DriveForwardCmd() [1/2]**

```
AutoCommand * TankDrive:::DriveForwardCmd (
    double dist,
    vex::directionType dir = vex::forward,
    double max_speed = 1.0,
    double end_speed = 0.0 )
```

**7.83.4.9 DriveForwardCmd() [2/2]**

```
AutoCommand * TankDrive:::DriveForwardCmd (
    Feedback & fb,
    double dist,
    vex::directionType dir = vex::forward,
    double max_speed = 1.0,
    double end_speed = 0.0 )
```

**7.83.4.10 DriveToPointCmd() [1/2]**

```
AutoCommand * TankDrive:::DriveToPointCmd (
    Feedback & fb,
    point_t pt,
    vex::directionType dir = vex::forward,
    double max_speed = 1.0,
    double end_speed = 0.0 )
```

**7.83.4.11 DriveToPointCmd() [2/2]**

```
AutoCommand * TankDrive:::DriveToPointCmd (
    point_t pt,
    vex::directionType dir = vex::forward,
    double max_speed = 1.0,
    double end_speed = 0.0 )
```

#### 7.83.4.12 `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

**7.83.4.13 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

**7.83.4.14 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

#### 7.83.4.15 PurePursuitCmd() [1/2]

```
AutoCommand * TankDrive::PurePursuitCmd (
    Feedback & feedback,
    PurePursuit::Path path,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

#### 7.83.4.16 PurePursuitCmd() [2/2]

```
AutoCommand * TankDrive::PurePursuitCmd (
    PurePursuit::Path path,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

#### 7.83.4.17 reset\_auto()

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

Reset the initialization for autonomous drive functions

#### 7.83.4.18 stop()

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

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

#### 7.83.4.19 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

### 7.83.4.20 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

#### 7.83.4.21 turn\_to\_heading() [1/2]

```
bool TankDrive::turn_to_heading (
    double heading_deg,
    double max_speed = 1,
    double end_speed = 0 )
```

Turn the robot in place to an exact heading relative to the field. 0 is forward. Uses the defualt turn feedback of the drive system

##### Parameters

<i>heading_deg</i>	the heading to which we will turn
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Turn the robot in place to an exact heading relative to the field. 0 is forward. Uses the defualt turn feedback of the drive system

##### Parameters

<i>heading_deg</i>	the heading to which we will turn
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

##### Returns

true if we have reached our target heading

#### 7.83.4.22 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

**7.83.4.23 TurnDegreesCmd() [1/2]**

```
AutoCommand * TankDrive::TurnDegreesCmd (
    double degrees,
    double max_speed = 1.0,
    double start_speed = 0.0 )
```

**7.83.4.24 TurnDegreesCmd() [2/2]**

```
AutoCommand * TankDrive::TurnDegreesCmd (
    Feedback & fb,
    double degrees,
    double max_speed = 1.0,
    double end_speed = 0.0 )
```

**7.83.4.25 TurnToHeadingCmd() [1/2]**

```
AutoCommand * TankDrive::TurnToHeadingCmd (
    double heading,
    double max_speed = 1.0,
    double end_speed = 0.0 )
```

**7.83.4.26 TurnToHeadingCmd() [2/2]**

```
AutoCommand * TankDrive::TurnToHeadingCmd (
    Feedback & fb,
    double heading,
    double max_speed = 1.0,
    double end_speed = 0.0 )
```

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

- include/subsystems/[tank\\_drive.h](#)
- src/subsystems/[tank\\_drive.cpp](#)

## 7.84 screen::TextConfig Struct Reference

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

### Public Attributes

- std::function< std::string()> [text](#)

### 7.84.1 Member Data Documentation

#### 7.84.1.1 [text](#)

```
std::function<std::string()> screen::TextConfig::text
```

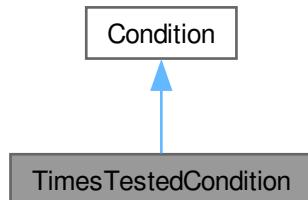
The documentation for this struct was generated from the following file:

- include/subsystems/[screen.h](#)

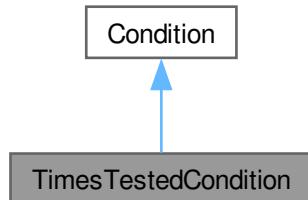
## 7.85 TimesTestedCondition Class Reference

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

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`)

### 7.85.1 Constructor & Destructor Documentation

#### 7.85.1.1 [TimesTestedCondition\(\)](#)

```
TimesTestedCondition::TimesTestedCondition (
    size_t N ) [inline]
```

### 7.85.2 Member Function Documentation

#### 7.85.2.1 [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`

## 7.86 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*

### 7.86.1 Detailed Description

`trapezoid_profile_segment_t` is a description of one constant acceleration segment of a trapezoid motion profile

### 7.86.2 Member Data Documentation

#### 7.86.2.1 accel

```
double trapezoid_profile_segment_t::accel
```

1d acceleration during the segment

#### 7.86.2.2 duration

```
double trapezoid_profile_segment_t::duration
```

duration of the segment

#### 7.86.2.3 pos\_after

```
double trapezoid_profile_segment_t::pos_after
```

1d position after this segment concludes

#### 7.86.2.4 vel\_after

```
double trapezoid_profile_segment_t::vel_after
```

1d velocity after this segment concludes

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

- include/utils/controls/[trapezoid\\_profile.h](#)

## 7.87 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`

### 7.87.1 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

## 7.87.2 Constructor & Destructor Documentation

### 7.87.2.1 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

## 7.87.3 Member Function Documentation

### 7.87.3.1 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_s</i>	Time since start of movement
<i>pos_s</i>	The current position

#### Returns

*motion\_t* Position, velocity and acceleration

### 7.87.3.2 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_s</i>	Time since start of movement
---------------	------------------------------

**Returns**

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

**7.87.3.3 get\_accel()**

```
double TrapezoidProfile::get_accel () const
```

**7.87.3.4 get\_max\_v()**

```
double TrapezoidProfile::get_max_v () const
```

**7.87.3.5 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

**7.87.3.6 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
--------------	--------------------------------

**7.87.3.7 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

### 7.87.3.8 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
--------------	--

### 7.87.3.9 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

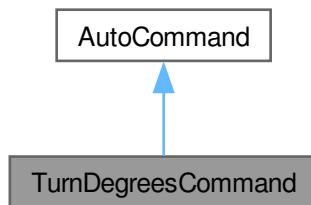
The documentation for this class was generated from the following files:

- [include/utils/controls/trapezoid\\_profile.h](#)
- [src/utils/trapezoid\\_profile.cpp](#)

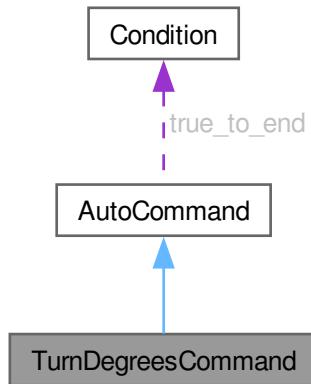
## 7.88 TurnDegreesCommand Class Reference

```
#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`

## 7.88.1 Detailed Description

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

## 7.88.2 Constructor & Destructor Documentation

### 7.88.2.1 TurnDegreesCommand()

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

Construct a [TurnDegreesCommand](#) Command

#### Parameters

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

## 7.88.3 Member Function Documentation

### 7.88.3.1 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](#).

### 7.88.3.2 run()

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

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

#### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

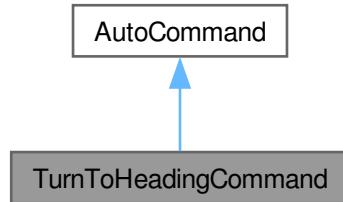
The documentation for this class was generated from the following files:

- include/utils/command\_structure/[drive\\_commands.h](#)
- src/utils/command\_structure/[drive\\_commands.cpp](#)

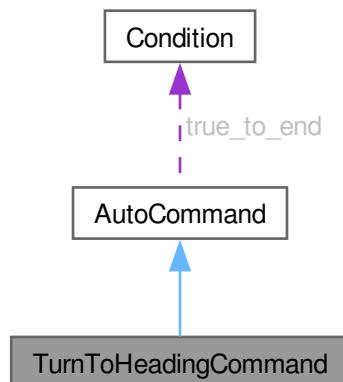
## 7.89 TurnToHeadingCommand Class Reference

```
#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

## 7.89.1 Detailed Description

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

## 7.89.2 Constructor & Destructor Documentation

### 7.89.2.1 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

## 7.89.3 Member Function Documentation

### 7.89.3.1 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](#).

### 7.89.3.2 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](#)

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

### 7.90.1 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

### 7.90.2 Constructor & Destructor Documentation

#### 7.90.2.1 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.

**7.90.2.2 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
----------	-----------------------

**7.90.3 Member Function Documentation****7.90.3.1 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.

**7.90.3.2 get\_mag()**

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

**Returns**

the magnitude of the vector

Get the magnitude of the vector

### 7.90.3.3 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.

### 7.90.3.4 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.

### 7.90.3.5 normalize()

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

Changes the magnitude of the vector to 1

#### Returns

the normalized vector

Changes the magnetude of the vector to 1

### 7.90.3.6 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

### 7.90.3.7 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

### 7.90.3.8 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

### 7.90.3.9 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:

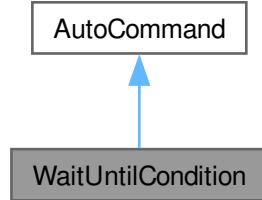
- include/utils/[vector2d.h](#)
- src/utils/[vector2d.cpp](#)

## 7.91 WaitUntilCondition Class Reference

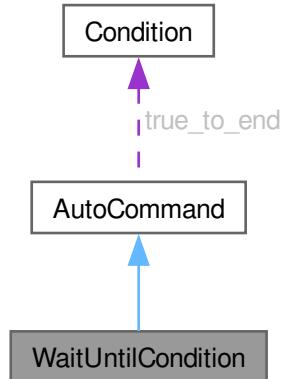
Waits until the condition is true.

```
#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

## 7.91.1 Detailed Description

Waits until the condition is true.

## 7.91.2 Constructor & Destructor Documentation

### 7.91.2.1 `WaitUntilCondition()`

```
WaitUntilCondition::WaitUntilCondition (
    Condition * cond ) [inline]
```

## 7.91.3 Member Function Documentation

### 7.91.3.1 `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](#).

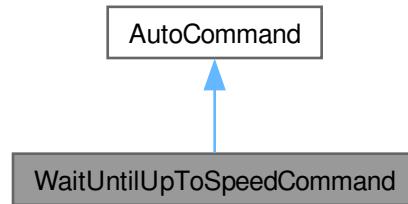
The documentation for this class was generated from the following file:

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

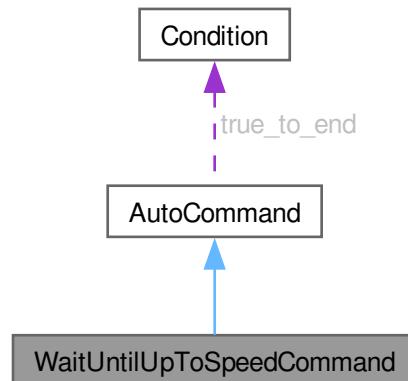
## 7.92 WaitUntilUpToSpeedCommand Class Reference

```
#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

## 7.92.1 Detailed Description

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

## 7.92.2 Constructor & Destructor Documentation

### 7.92.2.1 [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

## 7.92.3 Member Function Documentation

### 7.92.3.1 [run\(\)](#)

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

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

#### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

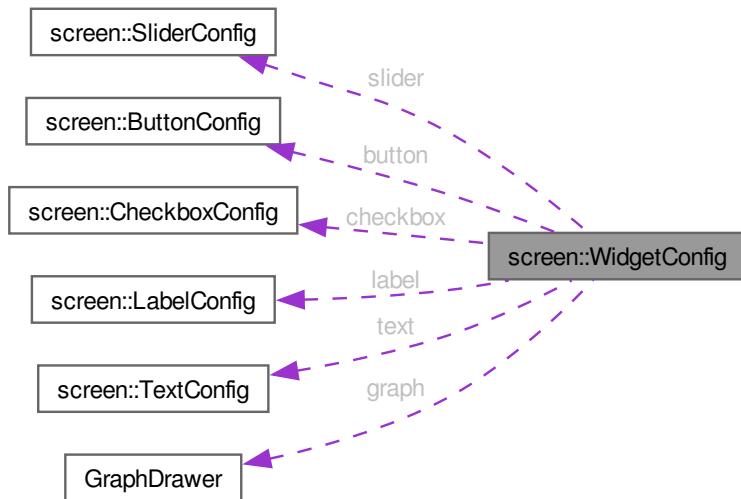
The documentation for this class was generated from the following files:

- include/utils/command\_structure/[flywheel\\_commands.h](#)
- src/utils/command\_structure/[flywheel\\_commands.cpp](#)

## 7.93 screen::WidgetConfig Struct Reference

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

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`

### 7.93.1 Member Enumeration Documentation

#### 7.93.1.1 Type

```
enum screen::WidgetConfig::Type
```

**Enumerator**

Col	
Row	
Slider	
Button	
Checkbox	
Label	
Text	
Graph	

## 7.93.2 Member Data Documentation

### 7.93.2.1 button

```
ButtonConfig screen::WidgetConfig::button
```

### 7.93.2.2 checkbox

```
CheckboxConfig screen::WidgetConfig::checkbox
```

### 7.93.2.3 [union]

```
union { ... } screen::WidgetConfig::config
```

### 7.93.2.4 graph

```
GraphDrawer* screen::WidgetConfig::graph
```

### 7.93.2.5 label

```
LabelConfig screen::WidgetConfig::label
```

### 7.93.2.6 slider

```
SliderConfig screen::WidgetConfig::slider
```

### 7.93.2.7 text

```
TextConfig screen::WidgetConfig::text
```

### 7.93.2.8 type

Type `screen::WidgetConfig::type`

### 7.93.2.9 widgets

`std::vector<SizedWidget> screen::WidgetConfig::widgets`

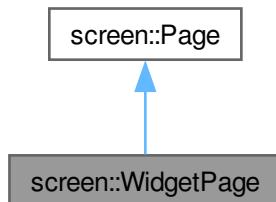
The documentation for this struct was generated from the following file:

- `include/subsystems/screen.h`

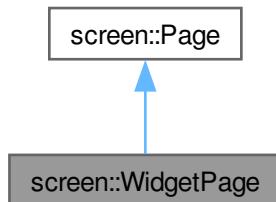
## 7.94 screen::WidgetPage Class Reference

`#include <screen.h>`

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)`

### 7.94.1 Constructor & Destructor Documentation

#### 7.94.1.1 WidgetPage()

```
screen::WidgetPage::WidgetPage (
    WidgetConfig & cfg ) [inline]
```

### 7.94.2 Member Function Documentation

#### 7.94.2.1 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

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

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

#### 7.94.2.2 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

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

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

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

- include/subsystems/[screen.h](#)

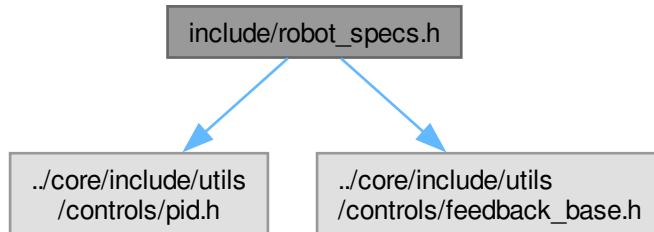


# Chapter 8

## File Documentation

### 8.1 include/robot\_specs.h File Reference

```
#include "../core/include/utils/controls/pid.h"
#include "../core/include/utils/controls/feedback_base.h"
Include dependency graph for robot_specs.h:
```



#### Classes

- struct [robot\\_specs\\_t](#)

### 8.2 robot\_specs.h

[Go to the documentation of this file.](#)

```
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;
```

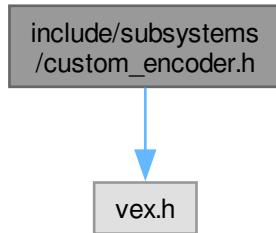
```

00017 double dist_between_wheels;
00018 double drive_correction_cutoff;
00019 Feedback *drive_feedback;
00020 Feedback *turn_feedback;
00021 PID::pid_config_t correction_pid;
00022
00023 } robot_specs_t;

```

## 8.3 include/subsystems/custom\_encoder.h File Reference

#include "vex.h"  
Include dependency graph for custom\_encoder.h:



### Classes

- class [CustomEncoder](#)

## 8.4 custom\_encoder.h

[Go to the documentation of this file.](#)

```

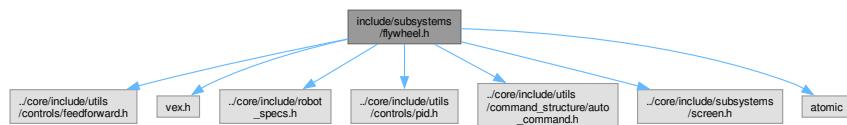
00001 #pragma once
00002 #include "vex.h"
00003
00008 class CustomEncoder : public vex::encoder
00009 {
00010     typedef vex::encoder super;
00011
00012 public:
00013     CustomEncoder(vex::tripoint::port &port, double ticks_per_rev);
00019
00025     void setRotation(double val, vex::rotationUnits units);
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 };

```

## 8.5 include/subsystems/flywheel.h File Reference

```
#include "../core/include/utils/controls/feedforward.h"
#include "vex.h"
#include "../core/include/robot_specs.h"
#include "../core/include/utils/controls/pid.h"
#include "../core/include/utils/command_structure/auto_command.h"
#include "../core/include/subsystems/screen.h"
#include <atomic>
```

Include dependency graph for flywheel.h:



### Classes

- class [Flywheel](#)

## 8.6 flywheel.h

[Go to the documentation of this file.](#)

```
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>
0010
0018 class Flywheel
0019 {
0020
0021 public:
0022     // CONSTRUCTORS, GETTERS, AND SETTERS
0031     Flywheel(vex::motor_group &motors, Feedback &feedback, FeedForward &helper, const double ratio,
0032               Filter &filt);
0033
0037     double get_target() const;
0038
0042     double getRPM() const;
0043
0047     vex::motor_group &get_motors() const;
0048
0055     void spin_manual(double speed, directionType dir = fwd);
0056
0062     void spin_rpm(double rpm);
0063
0067     void stop();
0068
0073     bool is_on_target()
0074     {
0075         return fb.is_on_target();
0076     }
0077
0082     screen::Page *Page() const;
0083
0089     AutoCommand *SpinRpmCmd(int rpm)
0090     {
0092         return new FunctionCommand([this, rpm]()
  
```

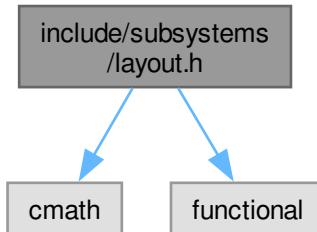
```

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 &ff;
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
00122     void set_target(double value);
00130     double measure_RPM();
00131
00138     void spin_raw(double speed, directionType dir = fwd);
00139 };

```

## 8.7 include/subsystems/layout.h File Reference

```
#include <cmath>
#include <functional>
Include dependency graph for layout.h:
```



### Classes

- struct SliderCfg

## 8.8 layout.h

[Go to the documentation of this file.](#)

```
00001 #include <cmath>
00002 #include <functional>
00003
```

```

00004 struct SliderCfg{
00005     double &val;
00006     double min;
00007     double max;
00008 };
00009
00010
00011

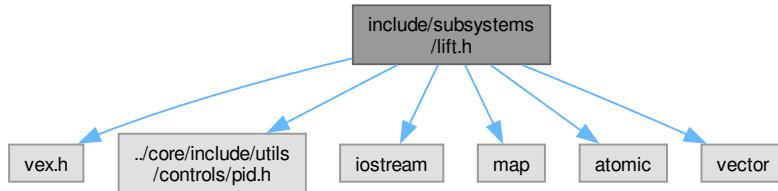
```

## 8.9 include/subsystems/lift.h File Reference

```

#include "vex.h"
#include "../core/include/utils/controls/pid.h"
#include <iostream>
#include <map>
#include <atomic>
#include <vector>
Include dependency graph for lift.h:

```



### Classes

- class [Lift< T >](#)
- struct [Lift< T >::lift\\_cfg\\_t](#)

## 8.10 lift.h

[Go to the documentation of this file.](#)

```

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     *homming_switch=NULL)
00061     : lift_motors(lift_motors), cfg(lift_cfg), lift_pid(cfg.lift_pid_cfg), setpoint_map(setpoint_map),
00061     homming_switch(homming_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         // Disable the PID while going UP.
00113         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         is_async = true;
00121     } else
00122     {
00123         // Hold the lift at the last setpoint
00124         is_async = true;
00125     }
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 }
00165
00177 void control_setpoints(bool up_step, bool down_step, vector<T> pos_list)
00178 {
00179     // Make sure inputs are only processed on the rising edge of the button
00180     static bool up_last = up_step, down_last = down_step;
00181
00182     bool up_rising = up_step && !up_last;
00183     bool down_rising = down_step && !down_last;
00184
00185     up_last = up_step;
00186     down_last = down_step;
00187
00188     static int cur_index = 0;
00189
00190     // Avoid an index overflow. Shouldn't happen unless the user changes pos_list between calls.
00191     if(cur_index >= pos_list.size())
00192         cur_index = pos_list.size() - 1;
00193
00194     // Increment or decrement the index of the list, bringing it up or down.
00195     if(up_rising && cur_index < (pos_list.size() - 1))
00196         cur_index++;
00197     else if(down_rising && cur_index > 0)
00198         cur_index--;
00199
00200     // Set the lift to hold the position in the background with the PID loop
00201     set_position(pos_list[cur_index]);
00202     is_async = true;
00203 }
00204
00205
00214 bool set_position(T pos)
00215 {
00216     this->setpoint = setpoint_map[pos];
00217     is_async = true;
00218
00219     return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
00220 }
00221
00228 bool set_setpoint(double val)
00229 {
00230     this->setpoint = val;
00231     return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
00232 }
00233
00237 double get_setpoint()
00238 {
00239     return this->setpoint;
00240 }
00241
00246 void hold()
00247 {
00248     lift_pid.set_target(setpoint);
00249     // std::cout << "DEBUG OUT: SETPOINT " << setpoint << "\n";
00250
00251     if(get_sensor != NULL)
00252         lift_pid.update(get_sensor());
00253     else
00254         lift_pid.update(lift_motors.position(rev));
00255
00256     // std::cout << "DEBUG OUT: ROTATION " << lift_motors.rotation(rev) << "\n\n";
00257
00258     lift_motors.spin(fwd, lift_pid.get(), volt);
00259 }
00260
00265 void home()
00266 {
00267     static timer tmr;
00268     tmr.reset();
00269
00270     while(tmr.time(sec) < 3)
00271     {
00272         lift_motors.spin(directionType::rev, 6, volt);
00273
00274         if (homing_switch == NULL && lift_motors.current(currentUnits::amp) > 1.5)
00275             break;
00276         else if (homing_switch != NULL && homing_switch->pressing())
00277             break;
00278     }
}

```

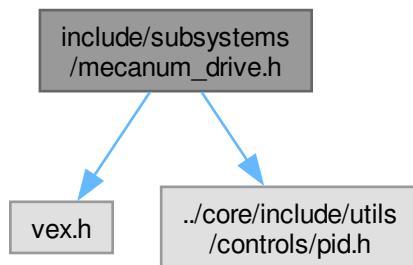
```

00279
00280     if(reset_sensor != NULL)
00281         reset_sensor();
00282
00283     lift_motors.resetPosition();
00284     lift_motors.stop();
00285
00286 }
00287
00288 bool get_async()
00289 {
00290     return is_async;
00291 }
00292
00293 void set_async(bool val)
00294 {
00295     this->is_async = val;
00296 }
00297
00298 void set_sensor_function(double (*fn_ptr) (void))
00299 {
00300     this->get_sensor = fn_ptr;
00301 }
00302
00303 void set_sensor_reset(void (*fn_ptr) (void))
00304 {
00305     this->reset_sensor = fn_ptr;
00306 }
00307
00308 private:
00309
00310     motor_group &lift_motors;
00311     lift_cfg_t &cfg;
00312     PID lift_pid;
00313     map<T, double> &setpoint_map;
00314     limit *homing_switch;
00315
00316     atomic<double> setpoint;
00317     atomic<bool> is_async;
00318
00319     double (*get_sensor) (void) = NULL;
00320     void (*reset_sensor) (void) = NULL;
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 };

```

## 8.11 include/subsystems/mecanum\_drive.h File Reference

```
#include "vex.h"
#include "../core/include/utils/controls/pid.h"
Include dependency graph for mecanum_drive.h:
```



## Classes

- class [MecanumDrive](#)
- struct [MecanumDrive::mecanumdrive\\_config\\_t](#)

## Macros

- `#define PI 3.141592654`

### 8.11.1 Macro Definition Documentation

#### 8.11.1.1 PI

```
#define PI 3.141592654
```

## 8.12 mecanum\_drive.h

[Go to the documentation of this file.](#)

```
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
00017     public:
00018
00022     struct mecanumdrive_config_t
00023     {
00024         // PID configurations for autonomous driving
00025         PID::pid_config_t drive_pid_conf;
00026         PID::pid_config_t drive_gyro_pid_conf;
00027         PID::pid_config_t turn_pid_conf;
00028
00029         // Diameter of the mecanum wheels
00030         double drive_wheel_diam;
00031
00032         // Diameter of the perpendicular undriven encoder wheel
00033         double lateral_wheel_diam;
00034
00035         // Width between the center of the left and right wheels
00036         double wheelbase_width;
00037
00038     };
00039
00043     MecanumDrive(vex::motor &left_front, vex::motor &right_front, vex::motor &left_rear, vex::motor
&right_rear,
00044             vex::rotation *lateral_wheel=NULL, vex::inertial *imu=NULL, mecanumdrive_config_t
*config=NULL);
00045
00054     void drive_raw(double direction_deg, double magnitude, double rotation);
00055
00066     void drive(double left_y, double left_x, double right_x, int power=2);
00067
00080     bool auto_drive(double inches, double direction, double speed, bool gyro_correction=true);
00081
00092     bool auto_turn(double degrees, double speed, bool ignore_imu=false);
00093
00094     private:
00095
00096     vex::motor &left_front, &right_front, &left_rear, &right_rear;
00097
00098     mecanumdrive_config_t *config;
00099     vex::rotation *lateral_wheel;
```

```

00100 vex::inertial *imu;
00101
00102 PID *drive_pid = NULL;
00103 PID *drive_gyro_pid = NULL;
00104 PID *turn_pid = NULL;
00105
00106 bool init = true;
00107
00108 };

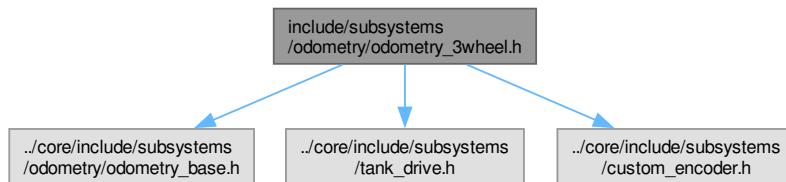
```

## 8.13 include/subsystems/odometry/odometry\_3wheel.h File Reference

```

#include "../core/include/subsystems/odometry/odometry_base.h"
#include "../core/include/subsystems/tank_drive.h"
#include "../core/include/subsystems/custom_encoder.h"
Include dependency graph for odometry_3wheel.h:

```



### Classes

- class [Odometry3Wheel](#)
- struct [Odometry3Wheel::odometry3wheel\\_cfg\\_t](#)

## 8.14 odometry\_3wheel.h

[Go to the documentation of this file.](#)

```

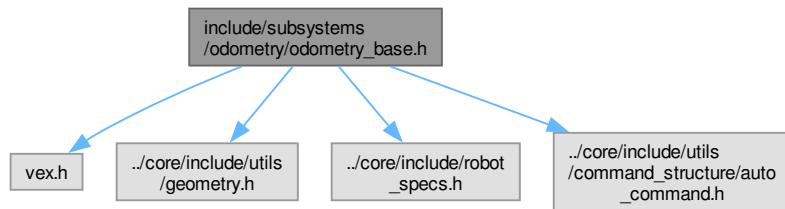
00001 #pragma once
00002 #include "../core/include/subsystems/odometry/odometry_base.h"
00003 #include "../core/include/subsystems/tank_drive.h"
00004 #include "../core/include/subsystems/custom_encoder.h"
00005
00032 class Odometry3Wheel : public OdometryBase
00033 {
00034     public:
00035     typedef struct
00041     {
00042         double wheelbase_dist;
00043         double off_axis_center_dist;
00044         double wheel_diam;
00046     } odometry3wheel_cfg_t;
00047
00057     Odometry3Wheel(CustomEncoder &lside_fwd, CustomEncoder &rside_fwd, CustomEncoder &off_axis,
00058     odometry3wheel_cfg_t &cfg, bool is_async=true);
00058
00065     pose_t update() override;
00066
00075     void tune(vex::controller &con, TankDrive &drive);
00076
00077     private:
00078

```

```
00091     static pose_t calculate_new_pos(double lside_delta_deg, double rside_delta_deg, double
00092         offax_delta_deg, pose_t old_pos, odometry3wheel_cfg_t cfg);
00093     CustomEncoder &lside_fwd, &rside_fwd, &off_axis;
00094     odometry3wheel_cfg_t &cfg;
00095
00096
00097 };
```

## 8.15 include/subsystems/odometry/odometry\_base.h File Reference

```
#include "vex.h"
#include "../core/include/utils/geometry.h"
#include "../core/include/robot_specs.h"
#include "../core/include/utils/command_structure/auto_command.h"
Include dependency graph for odometry_base.h:
```



### Classes

- class [OdometryBase](#)

### Macros

- #define [PI](#) 3.141592654

### 8.15.1 Macro Definition Documentation

#### 8.15.1.1 PI

```
#define PI 3.141592654
```

## 8.16 odometry\_base.h

[Go to the documentation of this file.](#)

```

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 };

```

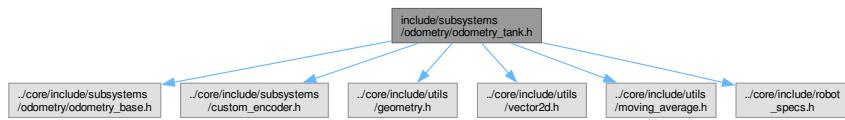
## 8.17 include/subsystems/odometry/odometry\_tank.h File Reference

```

#include "../core/include/subsystems/odometry/odometry_base.h"
#include "../core/include/subsystems/custom_encoder.h"
#include "../core/include/utils/geometry.h"
#include "../core/include/utils/vector2d.h"
#include "../core/include/utils/moving_average.h"
#include "../core/include/robot_specs.h"

```

Include dependency graph for odometry\_tank.h:



## Classes

- class [OdometryTank](#)

## 8.18 odometry\_tank.h

[Go to the documentation of this file.](#)

```

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 };
  
```

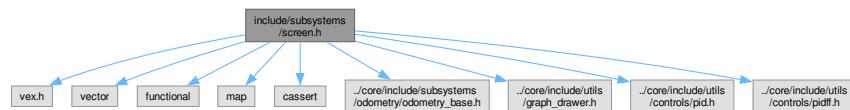
## 8.19 include/subsystems/screen.h File Reference

```

#include "vex.h"
#include <vector>
  
```

```
#include <functional>
#include <map>
#include <cassert>
#include "../core/include/subsystems/odometry/odometry_base.h"
#include "../core/include/utils/graph_drawer.h"
#include "../core/include/utils/controls/pid.h"
#include "../core/include/utils/controls/pidff.h"

Include dependency graph for screen.h:
```



## Classes

- class `screen::ButtonWidget`  
*Widget that does something when you tap it. The function is only called once when you first tap it.*
- class `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.*
- struct `screen::SliderConfig`
- struct `screen::ButtonConfig`
- struct `screen::CheckboxConfig`
- struct `screen::LabelConfig`
- struct `screen::TextConfig`
- struct `screen::SizedWidget`
- struct `screen::WidgetConfig`
- class `screen::Page`  
*Page describes one part of the screen slideshow.*
- struct `screen::ScreenRect`
- class `screen::WidgetPage`
- class `screen::StatsPage`  
*Draws motor stats and battery stats to the screen.*
- class `screen::OdometryPage`  
*a page that shows odometry position and rotation and a map (if an sd card with the file is on)*
- class `screen::FunctionPage`  
*Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything.*
- class `screen::PIDPage`  
*PIDPage provides a way to tune a pid controller on the screen.*

## Namespaces

- namespace `screen`

## Typedefs

- using `screen::update_func_t = std::function<void(bool, int, int)>`  
*type of function needed for update*
- using `screen::draw_func_t = std::function<void(vex::brain::lcd &screen, bool, unsigned int)>`  
*type of function needed for draw*

## Functions

- void `screen::draw_widget (WidgetConfig &widget, ScreenRect rect)`
- void `screen::start_screen (vex::brain::lcd &screen, std::vector< Page * > pages, int first_page=0)`  
*Start the screen background task. Once you start this, no need to draw to the screen manually elsewhere.*
- void `screen::next_page ()`
- void `screen::prev_page ()`
- void `screen::stop_screen ()`  
*stops the screen. If you have a drive team that hates fun call this at the start of opcontrol*

## 8.20 screen.h

[Go to the documentation of this file.](#)

```

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             bool update(bool was_pressed, int x, int y);
00030             void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
00031
00032         private:
00033             std::function<void(void)> onpress;
00034             Rect rect;
00035             std::string name = "";
00036             bool was_pressed_last = false;
00037         };
00038
00039     class SliderWidget
00040     {
00041         public:
00045             SliderWidget(double &val, double low, double high, Rect rect, std::string name) : value(val),
00046                 low(low), high(high), rect(rect), name(name) {}
00047
00048             bool update(bool was_pressed, int x, int y);
00049             void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
00050
00051         private:
00052             double &value;
00053
00054             double low;
00055             double high;
00056
00057             Rect rect;
00058             std::string name = "";
00059         };
00060
00061     struct WidgetConfig;
00062
00063     struct SliderConfig
00064     {
00065         double &val;
00066         double low;
00067         double high;
00068     };
00069
00070     struct ButtonConfig
00071     {
00072         std::function<void()> onclick;
00073     };
00074
00075     struct CheckboxConfig
00076
00077
00078
00079
00080
00081
00082
00083
00084
00085
00086
00087
00088

```

```

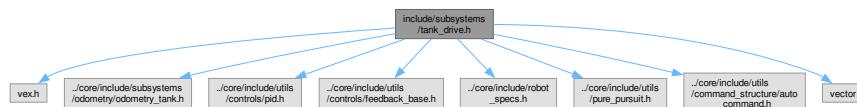
00089     {
00090         std::function<void(bool)> onupdate;
00091     };
00092     struct LabelConfig
00093     {
00094         std::string label;
00095     };
00096
00097     struct TextConfig
00098     {
00099         std::function<std::string()> text;
00100    };
00101     struct SizedWidget
00102    {
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:
00145         virtual void update(bool was_pressed, int x, int y);
00153         virtual void draw(vex::brain::lcd &screen, bool first_draw,
00154                         unsigned int frame_number);
00155     };
00156
00157     struct ScreenRect
00158     {
00159         uint32_t x1;
00160         uint32_t y1;
00161         uint32_t x2;
00162         uint32_t y2;
00163     };
00164     void draw_widget(WidgetConfig &widget, ScreenRect rect);
00165
00166     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
00187     void start_screen(vex::brain::lcd &screen, std::vector<Page *> pages, int first_page = 0);
00188
00189
00190     void next_page();
00191     void prev_page();
00192
00194     void stop_screen();
00195
00197     using update_func_t = std::function<void(bool, int, int)>;
00198
00199     using draw_func_t = std::function<void(vex::brain::lcd &screen, bool, unsigned int)>;

```

```
00201
00203     class StatsPage : public Page
00204     {
00205         public:
00206             StatsPage(std::map<std::string, vex::motor &> motors);
00207             void update(bool was_pressed, int x, int y) override;
00208             void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00209
00210         private:
00211             void draw_motor_stats(const std::string &name, vex::motor &mot, unsigned int frame, int x, int
00212 y, vex::brain::lcd &scr);
00213
00214     };
00215
00216
00217     class OdometryPage : public Page
00218     {
00219         public:
00220             OdometryPage(OdometryBase &odom, double robot_width, double robot_height, bool do_trail);
00221             void update(bool was_pressed, int x, int y) override;
00222             void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00223
00224         private:
00225             static const int path_len = 40;
00226             static constexpr char const *field_filename = "vex_field_240p.png";
00227
00228             OdometryBase &odom;
00229             double robot_width;
00230             double robot_height;
00231             uint8_t *buf = nullptr;
00232             int buf_size = 0;
00233             pose_t path[path_len];
00234             int path_index = 0;
00235             bool do_trail;
00236             GraphDrawer velocity_graph;
00237
00238
00239     class FunctionPage : public Page
00240     {
00241         public:
00242             FunctionPage(update_func_t update_f, draw_func_t draw_t);
00243             void update(bool was_pressed, int x, int y) override;
00244             void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00245
00246         private:
00247             update_func_t update_f;
00248             draw_func_t draw_f;
00249
00250
00251     class PIDPage : public Page
00252     {
00253         public:
00254             PIDPage(
00255                 PID &pid, std::string name, std::function<void(void)> onchange = []() {});
00256             PIDPage(
00257                 PIDFF &pidff, std::string name, std::function<void(void)> onchange = []() {});
00258
00259             void update(bool was_pressed, int x, int y) override;
00260             void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00261
00262         private:
00263             void zero_d_f() { cfg.d = 0; }
00264             void zero_i_f() { cfg.i = 0; }
00265
00266             PID::pid_config_t &cfg;
00267             PID &pid;
00268             const std::string name;
00269             std::function<void(void)> onchange;
00270
00271             SliderWidget p_slider;
00272             SliderWidget i_slider;
00273             SliderWidget d_slider;
00274             ButtonWidget zero_i;
00275             ButtonWidget zero_d;
00276
00277             GraphDrawer graph;
00278
00279     };
00280
00281 }
```

## 8.21 include/subsystems/tank\_drive.h File Reference

```
#include "vex.h"
#include "../core/include/subsystems/odometry/odometry_tank.h"
#include "../core/include/utils/controls/pid.h"
#include "../core/include/utils/controls/feedback_base.h"
#include "../core/include/robot_specs.h"
#include "../core/include/utils/pure_pursuit.h"
#include "../core/include/utils/command_structure/auto_command.h"
#include <vector>
Include dependency graph for tank_drive.h:
```



### Classes

- class [TankDrive](#)

### Macros

- #define PI 3.141592654

### 8.21.1 Macro Definition Documentation

#### 8.21.1.1 PI

```
#define PI 3.141592654
```

## 8.22 tank\_drive.h

[Go to the documentation of this file.](#)

```
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:
```

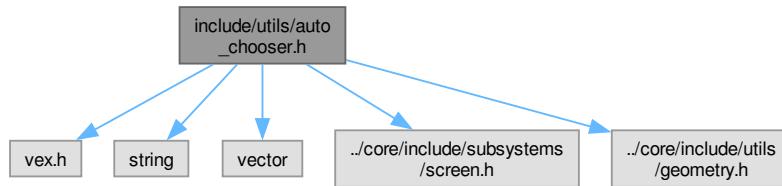
```

00025     enum class BrakeType
00026     {
00027         None,
00028         ZeroVelocity,
00029         Smart,
00030     };
00031     TankDrive(motor_group &left_motors, motor_group &right_motors, robot_specs_t &config, OdometryBase
00032     *odom = NULL);
00033
00034     AutoCommand *DriveToPointCmd(point_t pt, vex::directionType dir = vex::forward, double max_speed =
00035     1.0, double end_speed = 0.0);
00036     AutoCommand *DriveToPointCmd(Feedback &fb, point_t pt, vex::directionType dir = vex::forward, double
00037     max_speed = 1.0, double end_speed = 0.0);
00038
00039     AutoCommand *DriveForwardCmd(double dist, vex::directionType dir = vex::forward, double max_speed =
00040     1.0, double end_speed = 0.0);
00041     AutoCommand *DriveForwardCmd(Feedback &fb, double dist, vex::directionType dir = vex::forward,
00042     double max_speed = 1.0, double end_speed = 0.0);
00043
00044     AutoCommand *TurnToHeadingCmd(double heading, double max_speed = 1.0, double end_speed = 0.0);
00045     AutoCommand *TurnToHeadingCmd(Feedback &fb, double heading, double max_speed = 1.0, double end_speed
00046     = 0.0);
00047
00048     AutoCommand *TurnDegreesCmd(double degrees, double max_speed = 1.0, double start_speed = 0.0);
00049     AutoCommand *TurnDegreesCmd(Feedback &fb, double degrees, double max_speed = 1.0, double end_speed =
00050     0.0);
00051
00052     AutoCommand *PurePursuitCmd(PurePursuit::Path path, directionType dir, double max_speed = 1, double
00053     end_speed = 0);
00054     AutoCommand *PurePursuitCmd(Feedback &feedback, PurePursuit::Path path, directionType dir, double
00055     max_speed = 1, double end_speed = 0);
00056
00057     void stop();
00058
00059     void drive_tank(double left, double right, int power = 1, BrakeType bt = BrakeType::None);
00060     void drive_tank_raw(double left, double right);
00061
00062     void drive_arcade(double forward_back, double left_right, int power = 1, BrakeType bt =
00063     BrakeType::None);
00064
00065     bool drive_forward(double inches, directionType dir, Feedback &feedback, double max_speed = 1,
00066     double end_speed = 0);
00067
00068     bool drive_forward(double inches, directionType dir, double max_speed = 1, double end_speed = 0);
00069
00070     bool turn_degrees(double degrees, Feedback &feedback, double max_speed = 1, double end_speed = 0);
00071
00072     bool turn_degrees(double degrees, double max_speed = 1, double end_speed = 0);
00073
00074     bool drive_to_point(double x, double y, vex::directionType dir, Feedback &feedback, double max_speed
00075     = 1, double end_speed = 0);
00076
00077     bool drive_to_point(double x, double y, vex::directionType dir, double max_speed = 1, double
00078     end_speed = 0);
00079
00080     bool turn_to_heading(double heading_deg, Feedback &feedback, double max_speed = 1, double end_speed
00081     = 0);
00082     bool turn_to_heading(double heading_deg, double max_speed = 1, double end_speed = 0);
00083
00084     void reset_auto();
00085
00086     static double modify_inputs(double input, int power = 2);
00087
00088     bool pure_pursuit(PurePursuit::Path path, directionType dir, Feedback &feedback, double max_speed =
00089     1, double end_speed = 0);
00090
00091     bool pure_pursuit(PurePursuit::Path path, directionType dir, double max_speed = 1, double end_speed
00092     = 0);
00093
00094     private:
00095     motor_group &left_motors;
00096     motor_group &right_motors;
00097
00098     PID correction_pid;
00099     Feedback *drive_default_feedback = NULL;
00100     Feedback *turn_default_feedback = NULL;
00101
00102     OdometryBase *odometry;
00103
00104     robot_specs_t &config;
00105
00106     bool func_initialized = false;
00107     bool is_pure_pursuit = false;
00108 };

```

## 8.23 include/utils/auto\_chooser.h File Reference

```
#include "vex.h"
#include <string>
#include <vector>
#include "../core/include/subsystems/screen.h"
#include "../core/include/utils/geometry.h"
Include dependency graph for auto_chooser.h:
```



### Classes

- class [AutoChooser](#)
- struct [AutoChooser::entry\\_t](#)

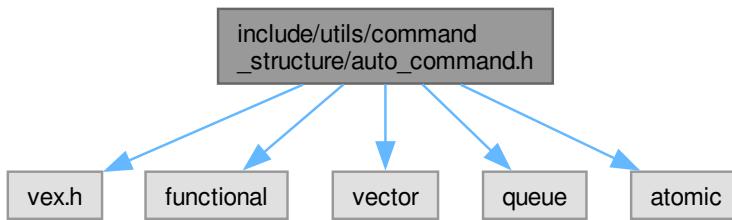
## 8.24 auto\_chooser.h

[Go to the documentation of this file.](#)

```
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 },
```

## 8.25 include/utils/command\_structure/auto\_command.h File Reference

```
#include "vex.h"
#include <functional>
#include <vector>
#include <queue>
#include <atomic>
Include dependency graph for auto_command.h:
```



### Classes

- class [Condition](#)
- class [AutoCommand](#)
- class [FunctionCommand](#)
- class [TimesTestedCondition](#)
- class [FunctionCondition](#)

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

*IfTimePassed* tests based on time since the command controller was constructed. Returns true if elapsed time > time\_s.
- class [WaitUntilCondition](#)

*Waits until the condition is true.*
- class [InOrder](#)

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

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

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

## 8.26 auto\_command.h

[Go to the documentation of this file.](#)

```

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 };
00118 class FunctionCondition : public Condition

```

```
0019 {
0020     public:
0021         FunctionCondition(
0022             std::function<bool()> cond, std::function<void(void)> timeout = []() {} ) : cond(cond),
0023             timeout(timeout)
0024     {
0025     }
0026     bool test() override;
0027
0028     private:
0029         std::function<bool()> cond;
0030         std::function<void(void)> timeout;
0031     };
0032
0033     class IfTimePassed : public Condition
0034     {
0035         public:
0036             IfTimePassed(double time_s);
0037             bool test() override;
0038
0039         private:
0040             double time_s;
0041             vex::timer tmr;
0042     };
0043
0044     class WaitUntilCondition : public AutoCommand
0045     {
0046         public:
0047             WaitUntilCondition(Condition *cond) : cond(cond) {}
0048             bool run() override
0049             {
0050                 return cond->test();
0051             }
0052         }
0053
0054         private:
0055             Condition *cond;
0056     };
0057
0058
0059     class InOrder : public AutoCommand
0060     {
0061         public:
0062             InOrder(const InOrder &other) = default;
0063             InOrder(std::queue<AutoCommand *> cmdqs);
0064             InOrder(std::initializer_list<AutoCommand *> cmdqs);
0065             bool run() override;
0066             void on_timeout() override;
0067
0068         private:
0069             AutoCommand *current_command = nullptr;
0070             std::queue<AutoCommand *> cmdqs;
0071             vex::timer tmr;
0072     };
0073
0074
0075     class Parallel : public AutoCommand
0076     {
0077         public:
0078             Parallel(std::initializer_list<AutoCommand *> cmdqs);
0079             bool run() override;
0080             void on_timeout() override;
0081
0082         private:
0083             std::vector<AutoCommand *> cmdqs;
0084             std::vector<vex::task *> runners;
0085     };
0086
0087
0088     class Branch : public AutoCommand
0089     {
0090         public:
0091             Branch(Condition *cond, AutoCommand *false_choice, AutoCommand *true_choice);
0092             ~Branch();
0093             bool run() override;
0094             void on_timeout() override;
0095
0096         private:
0097             AutoCommand *false_choice;
0098             AutoCommand *true_choice;
0099             Condition *cond;
0100             bool choice = false;
0101             bool chosen = false;
0102             vex::timer tmr;
0103     };
0104
0105
0106     class Async : public AutoCommand
0107     {
0108         public:
0109             Async(AutoCommand *cmd) : cmd(cmd) {}
0110
0111
0112
0113
0114
0115
0116
0117
0118
0119
0120
0121
0122
0123
0124
0125
0126
0127
0128
0129
0130
0131
0132
0133
0134
0135
0136
0137
0138
0139
0140
0141
0142
0143
0144
0145
0146
0147
0148
0149
0150
0151
0152
0153
0154
0155
0156
0157
0158
0159
0160
0161
0162
0163
0164
0165
0166
0167
0168
0169
0170
0171
0172
0173
0174
0175
0176
0177
0178
0179
0180
0181
0182
0183
0184
0185
0186
0187
0188
0189
0190
0191
0192
0193
0194
0195
0196
0197
0198
0199
0200
0201
0202
0203
0204
0205
0206
0207
0208
0209
0210
0211
0212
0213
0214
0215
0216
0217
0218
```

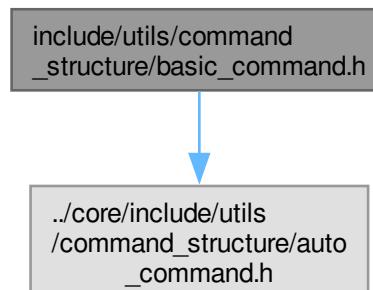
```

00219     bool run() override;
00220
00221 private:
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

```

## 8.27 include/utils/command\_structure/basic\_command.h File Reference

#include "../core/include/utils/command\_structure/auto\_command.h"  
Include dependency graph for basic\_command.h:



### Classes

- class BasicSpinCommand
- class BasicStopCommand
- class BasicSolenoidSet

## 8.28 basic\_command.h

[Go to the documentation of this file.](#)

```

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:

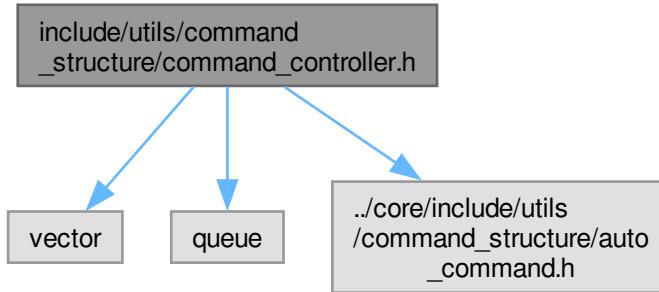
```

```
00026
00027 //Enumurato for the type of power setting in the motor
00028 enum type {percent,voltage,veocity};
00029
00030
00031     BasicSpinCommand(vex::motor &motor, vex::directionType dir, BasicSpinCommand::type setting,
00032     double power);
00033
00034
00035     bool run() override;
00036
00037 private:
00038
00039     vex::motor &motor;
00040
00041     type setting;
00042
00043     vex::directionType dir;
00044
00045     double power;
00046 };
00047
00048 class BasicStopCommand : public AutoCommand{
00049 public:
00050
00051     BasicStopCommand(vex::motor &motor, vex::brakeType setting);
00052
00053     bool run() override;
00054
00055 private:
00056
00057     vex::motor &motor;
00058
00059     vex::brakeType setting;
00060 };
00061
00062
00063 //Basic Solenoid Commands-----
00064
00065 class BasicSolenoidSet : public AutoCommand{
00066 public:
00067
00068     BasicSolenoidSet(vex::pneumatics &solenoid, bool setting);
00069
00070     bool run() override;
00071
00072 private:
00073
00074     vex::pneumatics &solenoid;
00075
00076     bool setting;
00077 };
00078
```

## 8.29 include/utils/command\_structure/command\_controller.h File Reference

```
#include <vector>
#include <queue>
#include "../core/include/utils/command_structure/auto_command.h"
```

Include dependency graph for command\_controller.h:



## Classes

- class [CommandController](#)

## 8.30 command\_controller.h

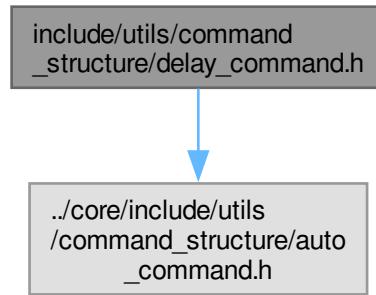
[Go to the documentation of this file.](#)

```

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:
00019   [[deprecated("Use list constructor instead.")]] CommandController() : command_queue({}) {}
00020
00023 CommandController(std::initializer_list<AutoCommand *> cmdqs) : command_queue(cmdqs) {}
00029   [[deprecated("Use list constructor instead. If you need to make a decision before adding new
commands, use Branch (https://github.com/RIT-VEX-U/Core/wiki/3-%7C-Utilites#commandcontroller)")]]
void add(std::vector<AutoCommand *> cmdqs);
00030   void add(AutoCommand *cmd, double timeout_seconds = 10.0);
00031
00042   [[deprecated("Use list constructor instead. If you need to make a decision before adding new
commands, use Branch (https://github.com/RIT-VEX-U/Core/wiki/3-%7C-Utilites#commandcontroller)")]]
void
00043   add(std::vector<AutoCommand *> cmdqs, double timeout_sec);
00050   void add_delay(int ms);
00051
00054   void add_cancel_func(std::function<bool(void)> true_if_cancel);
00055
00060   void run();
00061
00067   bool last_command_timed_out();
00068
00069 private:
00070   std::queue<AutoCommand *> command_queue;
00071   bool command_timed_out = false;
00072   std::function<bool()> should_cancel = []()
00073   { return false; };
00074 };
  
```

## 8.31 include/utils/command\_structure/delay\_command.h File Reference

```
#include "../core/include/utils/command_structure/auto_command.h"
Include dependency graph for delay_command.h:
```



### Classes

- class [DelayCommand](#)

## 8.32 delay\_command.h

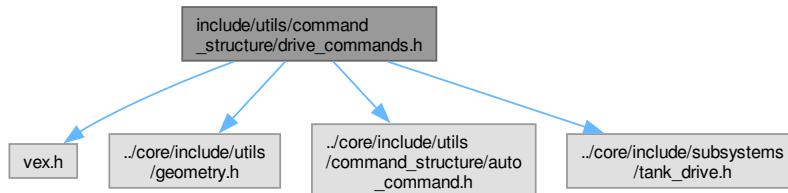
[Go to the documentation of this file.](#)

```
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     };
00025 }
```

## 8.33 include/utils/command\_structure/drive\_commands.h File Reference

```
#include "vex.h"
#include "../core/include/utils/geometry.h"
#include "../core/include/utils/command_structure/auto_command.h"
```

```
#include "../core/include/subsystems/tank_drive.h"
Include dependency graph for drive_commands.h:
```



## Classes

- class [DriveForwardCommand](#)
- class [TurnDegreesCommand](#)
- class [DriveToPointCommand](#)
- class [TurnToHeadingCommand](#)
- class [PurePursuitCommand](#)
- class [DriveStopCommand](#)
- class [OdomSetPosition](#)

## 8.34 drive\_commands.h

[Go to the documentation of this file.](#)

```

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
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
00046     bool run() override;
00050     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
00076     bool run() override;
00077     void on_timeout() override;
00078
00079
00080     private:
00081         // drive system to run the function on
00082         TankDrive &drive_sys;
00083
00084         // feedback controller to use
00085         Feedback &feedback;
00086
00087         // parameters for turn_degrees
00088         double degrees;
00089         double max_speed;
00090         double end_speed;
00091     };
00092
00093
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);
00109             DriveToPointCommand(TankDrive &drive_sys, Feedback &feedback, point_t point, directionType dir,
00110             double max_speed=1, double end_speed = 0);
00111
00112         bool run() override;
00113
00114         private:
00115             // drive system to run the function on
00116             TankDrive &drive_sys;
00117
00118             void on_timeout() override;
00119
00120             // feedback controller to use
00121             Feedback &feedback;
00122
00123             // parameters for drive_to_point
00124             double x;
00125             double y;
00126             directionType dir;
00127             double max_speed;
00128             double end_speed;
00129     };
00130
00131
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);
00149
00150             bool run() override;
00151             void on_timeout() override;
00152
00153
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
00174
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;
```

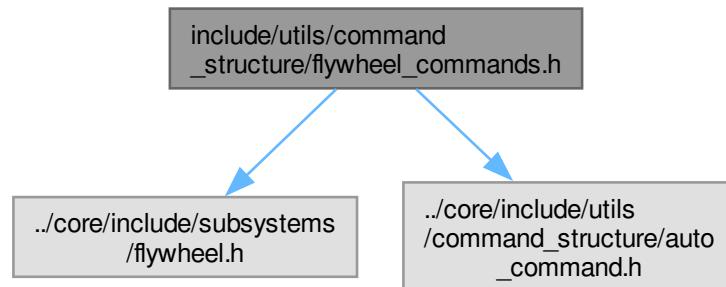
```

00205     double max_speed;
00206     double end_speed;
00207
00208 };
00209
00214 class DriveStopCommand: public AutoCommand
00215 {
00216     public:
00217     DriveStopCommand(TankDrive &drive_sys);
00218
00224     bool run() override;
00225     void on_timeout() override;
00226
00227     private:
00228     // drive system to run the function on
00229     TankDrive &drive_sys;
00230 };
00231
00232
00233 // ===== ODOMETRY =====
00234
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

```

## 8.35 include/utils/command\_structure/flywheel\_commands.h File Reference

```
#include "../core/include/subsystems/flywheel.h"
#include "../core/include/utils/command_structure/auto_command.h"
Include dependency graph for flywheel_commands.h:
```



### Classes

- class [SpinRPMCommand](#)
- class [WaitUntilUpToSpeedCommand](#)
- class [FlywheelStopCommand](#)
- class [FlywheelStopMotorsCommand](#)
- class [FlywheelStopNonTasksCommand](#)

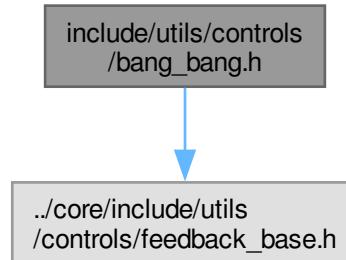
## 8.36 flywheel\_commands.h

[Go to the documentation of this file.](#)

```
00001
00007 #pragma once
00008
00009 #include "../core/include/subsystems/flywheel.h"
00010 #include "../core/include/utils/command_structure/auto_command.h"
00011
00017 class SpinRPMCommand: public AutoCommand {
00018     public:
00024     SpinRPMCommand(Flywheel &flywheel, int rpm);
00025
00031     bool run() override;
00032
00033     private:
00034         // Flywheel instance to run the function on
00035         Flywheel &flywheel;
00036
00037         // parameters for spin_rpm
00038         int rpm;
00039     };
00040
00045 class WaitUntilUpToSpeedCommand: public AutoCommand {
00046     public:
00052     WaitUntilUpToSpeedCommand(Flywheel &flywheel, int threshold_rpm);
00053
00059     bool run() override;
00060
00061     private:
00062         // Flywheel instance to run the function on
00063         Flywheel &flywheel;
00064
00065         // if the actual speed is equal to the desired speed +/- this value, we are ready to fire
00066         int threshold_rpm;
00067     };
00068
00074 class FlywheelStopCommand: public AutoCommand {
00075     public:
00080     FlywheelStopCommand(Flywheel &flywheel);
00081
00087     bool run() override;
00088
00089     private:
00090         // Flywheel instance to run the function on
00091         Flywheel &flywheel;
00092     };
00093
00099 class FlywheelStopMotorsCommand: public AutoCommand {
00100     public:
00105     FlywheelStopMotorsCommand(Flywheel &flywheel);
00106
00112     bool run() override;
00113
00114     private:
00115         // Flywheel instance to run the function on
00116         Flywheel &flywheel;
00117     };
00118
00124 class FlywheelStopNonTasksCommand: public AutoCommand {
00125     FlywheelStopNonTasksCommand(Flywheel &flywheel);
00126
00132     bool run() override;
00133
00134     private:
00135         // Flywheel instance to run the function on
00136         Flywheel &flywheel;
00137     };
```

## 8.37 include/utils/controls/bang\_bang.h File Reference

```
#include "../core/include/utils/controls/feedback_base.h"
Include dependency graph for bang_bang.h:
```



### Classes

- class [BangBang](#)

## 8.38 bang\_bang.h

[Go to the documentation of this file.](#)

```
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     };
00050 
```

## 8.39 include/utils/controls/feedback\_base.h File Reference

### Classes

- class [Feedback](#)

## 8.40 feedback\_base.h

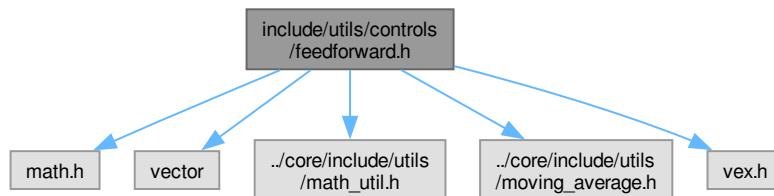
[Go to the documentation of this file.](#)

```
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 };
```

## 8.41 include/utils/controls/feedforward.h File Reference

```
#include <math.h>
#include <vector>
#include "../core/include/utils/math_util.h"
#include "../core/include/utils/moving_average.h"
#include "vex.h"
```

Include dependency graph for feedforward.h:



### Classes

- class [FeedForward](#)
- struct [FeedForward::ff\\_config\\_t](#)

### Functions

- [FeedForward::ff\\_config\\_t tune\\_feedforward](#) (vex::motor\_group &motor, double pct, double duration)

## 8.41.1 Function Documentation

### 8.41.1.1 tune\_feedforward()

```
FeedForward::ff_config_t tune_feedforward (
    vex::motor_group & motor,
    double pct,
    double duration )
```

tune\_feedforward takes a group of motors and finds the feedforward config parameters automagically.

**Parameters**

<i>motor</i>	the motor group to use
<i>pct</i>	Maximum velocity in percent (0->1.0)
<i>duration</i>	Amount of time the motors spin for the test

**Returns**

A tuned feedforward object

## 8.42 feedforward.h

[Go to the documentation of this file.](#)

```

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
00079     ff_config_t &cfg;
00080
00081 };
00082
00083
00091 FeedForward::ff_config_t tune_feedforward(vex::motor_group &motor, double pct, double duration);

```

## 8.43 include/utils/controls/motion\_controller.h File Reference

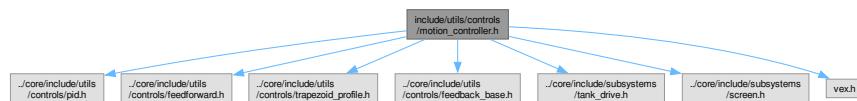
```

#include "../core/include/utils/controls/pid.h"
#include "../core/include/utils/controls/feedforward.h"
#include "../core/include/utils/controls/trapezoid_profile.h"
#include "../core/include/utils/controls/feedback_base.h"
#include "../core/include/subsystems/tank_drive.h"
#include "../core/include/subsystems/screen.h"

```

```
#include "vex.h"
```

Include dependency graph for motion\_controller.h:



## Classes

- class [MotionController](#)
- struct [MotionController::m\\_profile\\_cfg\\_t](#)

## 8.44 motion\_controller.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include "../core/include/utils/controls/pid.h"
00003 #include "../core/include/utils/controls/feedforward.h"
00004 #include "../core/include/utils/controls/trapezoid_profile.h"
00005 #include "../core/include/utils/controls/feedback_base.h"
00006 #include "../core/include/subsystems/tank_drive.h"
00007 #include "../core/include/subsystems/screen.h"
00008
00009 #include "vex.h"
00010
00027 class MotionController : public Feedback
00028 {
00029     public:
00030
00036     typedef struct
00037     {
00038         double max_v;
00039         double accel;
00040         PID::pid_config_t pid_cfg;
00041         FeedForward::ff_config_t ff_cfg;
00042     } m_profile_cfg_t;
00043
00053     MotionController(m_profile_cfg_t &config);
00054
00059     void init(double start_pt, double end_pt, double start_vel, double end_vel) override;
00060
00067     double update(double sensor_val) override;
00068
00072     double get() override;
00073
00081     void set_limits(double lower, double upper) override;
00082
00087     bool is_on_target() override;
00088
00092     motion_t get_motion() const;
00093
00094
00095     screen::Page *Page();
00096
00115     static FeedForward::ff_config_t tune_feedforward(TankDrive &drive, OdometryTank &odometry, double
      pct=0.6, double duration=2);
00116
00117     private:
00118
00119     m_profile_cfg_t config;
00120
00121     PID pid;
00122     FeedForward ff;
00123     TrapezoidProfile profile;
00124
00125     double current_pos;
00126     double end_pt;
00127
00128     double lower_limit = 0, upper_limit = 0;
  
```

```

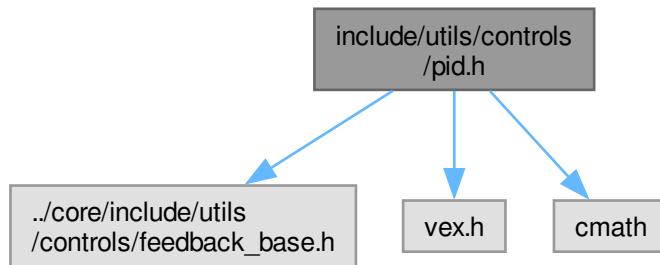
00129     double out = 0;
00130     motion_t cur_motion;
00131
00132     vex::timer tmr;
00133     friend class MotionControllerPage;
00134
00135 };

```

## 8.45 include/utils/controls/pid.h File Reference

```
#include "../core/include/utils/controls/feedback_base.h"
#include "vex.h"
#include <cmath>
```

Include dependency graph for pid.h:



### Classes

- class [PID](#)
- struct [PID::pid\\_config\\_t](#)

## 8.46 pid.h

[Go to the documentation of this file.](#)

```

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
00023 class PID : public Feedback {
00024 public:
00029     enum ERROR_TYPE {
00030         LINEAR,
00031         ANGULAR // assumes degrees
00032     };
00043     struct pid_config_t {
00044         double p;
00045         double i;
00046         double d;
00047         double deadband;
00048         double on_target_time;
00050         ERROR_TYPE error_method;

```

```

00052  };
00053
00054 PID(pid_config_t &config);
00055
00056 void init(double start_pt, double set_pt, double start_vel = 0,
00057             double end_vel = 0) override;
00058
00059 double update(double sensor_val) override;
00060
00061 double get_sensor_val() const;
00062
00063 double get() override;
00064
00065 void set_limits(double lower, double upper) override;
00066
00067 bool is_on_target() override;
00068
00069 void reset();
00070
00071 double get_error();
00072
00073 double get_target() const;
00074
00075 void set_target(double target);
00076
00077 pid_config_t
00078     &config;
00079
00080 private:
00081     double last_error =
00082         0;
00083     double accum_error =
00084         0;
00085
00086     double last_time = 0;
00087     double on_target_last_time =
00088         0;
00089
00090     double lower_limit =
00091         0;
00092     double upper_limit =
00093         0;
00094
00095     double target = 0;
00096     double target_vel = 0;
00097     double sensor_val = 0;
00098     double out = 0;
00099
00100     bool is_checking_on_target =
00101         false;
00102
00103     timer pid_timer;
00104 };

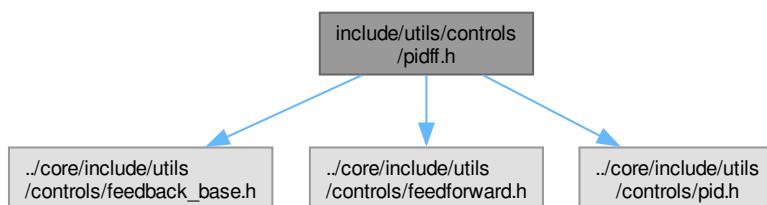
```

## 8.47 include/utils/controls/pidff.h File Reference

```

#include "../core/include/utils/controls/feedback_base.h"
#include "../core/include/utils/controls/feedforward.h"
#include "../core/include/utils/controls/pid.h"
Include dependency graph for pidff.h:

```



## Classes

- class [PIDFF](#)

## 8.48 pidff.h

[Go to the documentation of this file.](#)

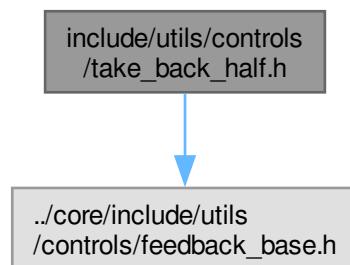
```

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
00010     void init(double start_pt, double set_pt, double start_vel,
00011               double end_vel) override;
00012
00013     void set_target(double set_pt);
00014
00015     double get_target() const;
00016     double get_sensor_val() const;
00017     double update(double val) override;
00018
00019     double update(double val, double vel_setpt, double a_setpt = 0);
00020
00021     double get() override;
00022
00023     void set_limits(double lower, double upper) override;
00024
00025     bool is_on_target() override;
00026
00027     void reset();
00028
00029     PID pid;
00030
00031 private:
00032     FeedForward::ff_config_t &ff_cfg;
00033
00034     FeedForward ff;
00035
00036     double out;
00037     double lower_lim, upper_lim;
00038 };

```

## 8.49 include/utils/controls/take\_back\_half.h File Reference

```
#include "../core/include/utils/controls/feedback_base.h"
Include dependency graph for take_back_half.h:
```



## Classes

- class [TakeBackHalf](#)

*A velocity controller.*

## 8.50 take\_back\_half.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "../core/include/utils/controls/feedback_base.h"
00003
00006 class TakeBackHalf : public Feedback
00007 {
00008
00009 public:
00010     TakeBackHalf(double TBH_gain, double first_cross_split, double on_target_threshold);
00019     void init(double start_pt, double set_pt, double, double);
00026     double update(double val) override;
00027
00031     double get() override;
00032
00039     void set_limits(double lower, double upper) override;
00040
00044     bool is_on_target() override;
00045
00046     double TBH_gain;
00047     double first_cross_split;
00048 private:
00049     double on_target_threshold;
00050
00051     double target = 0.0;
00052
00053     bool first_cross = true;
00054     double tbh = 0.0;
00055     double prev_error = 0.0;
00056
00057     double output = 0.0;
00058     double lower = 0.0, upper = 0.0;
00059 },
```

## 8.51 include/utils/controls/trapezoid\_profile.h File Reference

### Classes

- struct [motion\\_t](#)
- struct [trapezoid\\_profile\\_segment\\_t](#)
- class [TrapezoidProfile](#)

### Variables

- const int [MAX\\_TRAPEZOID\\_PROFILE\\_SEGMENTS](#) = 4

### 8.51.1 Variable Documentation

#### 8.51.1.1 MAX\_TRAPEZOID\_PROFILE\_SEGMENTS

```
const int MAX_TRAPEZOID_PROFILE_SEGMENTS = 4
```

## 8.52 trapezoid\_profile.h

[Go to the documentation of this file.](#)

```

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
00013
00014
00015
00016
00017
00018
00019 typedef struct {
00020     double pos_after;
00021     double vel_after;
00022     double accel;
00023     double duration;
00024 } trapezoid_profile_segment_t;
00025
00026
00027
00028
00029
00030
00031
00032
00033
00034
00035
00036
00037
00038
00039
00040
00041
00042
00043
00044
00045
00046
00047
00048
00049
00050
00051
00052
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063 class TrapezoidProfile {
00064 public:
00065     TrapezoidProfile(double max_v, double accel);
00066
00067     motion_t calculate(double time_s, double pos_s);
00068
00069     motion_t calculate_time_based(double time_s);
00070
00071     void set_endpts(double start, double end);
00072
00073     void set_vel_endpts(double start, double end);
00074
00075     void set_accel(double accel);
00076
00077     void set_max_v(double max_v);
00078
00079     double get_movement_time() const;
00080
00081     double get_max_v() const;
00082     double get_accel() const;
00083
00084 private:
00085     double si, sf;
00086     double vi, vf;
00087     double max_v;
00088     double accel;
00089     double duration;
00090
00091     trapezoid_profile_segment_t segments[MAX_TRAPEZOID_PROFILE_SEGMENTS];
00092     int num_acceleration_phases;
00093
00094     bool precalculated;
00095
00096     bool precalculate();
00097
00098     trapezoid_profile_segment_t calculate_kinetic_motion(double si, double vi,
00099                                         double v_target);
00100
00101     trapezoid_profile_segment_t calculate_next_segment(double s, double v);
00102 };

```

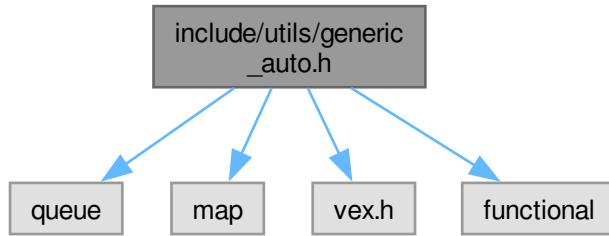
## 8.53 include/utils/generic\_auto.h File Reference

```

#include <queue>
#include <map>
#include "vex.h"
#include <functional>

```

Include dependency graph for generic\_auto.h:



## Classes

- class [GenericAuto](#)

## TypeDefs

- typedef std::function< bool(void)> [state\\_ptr](#)

### 8.53.1 TypeDef Documentation

#### 8.53.1.1 state\_ptr

```
typedef std::function<bool (void)> state_ptr
```

## 8.54 generic\_auto.h

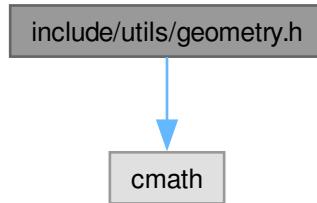
[Go to the documentation of this file.](#)

```

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 };
  
```

## 8.55 include/utils/geometry.h File Reference

```
#include <cmath>
Include dependency graph for geometry.h:
```



### Classes

- struct `point_t`
- struct `pose_t`
- struct `Rect`
- struct `Mat2`

## 8.56 geometry.h

[Go to the documentation of this file.](#)

```
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     {
```

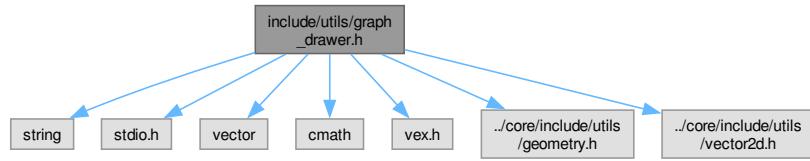
```

00054         return {x / s, y / s};
00055     }
00056
00057     point_t operator-() const
00058     {
00059         return {-x, -y};
00060     }
00061     point_t operator+() const
00062     {
00063         return {x, y};
00064     }
00065
00066     bool operator==(const point_t &rhs)
00067     {
00068         return x == rhs.x && y == rhs.y;
00069     }
00070 };
00071
00075 struct pose_t
00076 {
00077     double x;
00078     double y;
00079     double rot;
00080
00081     point_t get_point()
00082     {
00083         return point_t{.x = x, .y = y};
00084     }
00085
00086 } ;
00087
00088 struct Rect
00089 {
00090     point_t min;
00091     point_t max;
00092     static Rect from_min_and_size(point_t min, point_t size){
00093         return {min, min+size};
00094     }
00095     point_t dimensions() const
00096     {
00097         return max - min;
00098     }
00099     point_t center() const{
00100         return (min + max)/2;
00101     }
00102     double width() const{
00103         return max.x - min.x;
00104     }
00105     double height() const{
00106         return max.y - min.y;
00107     }
00108     bool contains(point_t p) const
00109     {
00110         bool xin = p.x > min.x && p.x < max.x;
00111         bool yin = p.y > min.y && p.y < max.y;
00112         return xin && yin;
00113     }
00114
00115 };
00116
00117 struct Mat2
00118 {
00119     double X11, X12;
00120     double X21, X22;
00121     point_t operator*(const point_t p) const
00122     {
00123         double outx = p.x * X11 + p.y * X12;
00124         double outy = p.x * X21 + p.y * X22;
00125         return {outx, outy};
00126     }
00127
00128     static Mat2 FromRotationDegrees(double degrees)
00129     {
00130         double rad = degrees * (M_PI / 180.0);
00131         double c = cos(rad);
00132         double s = sin(rad);
00133         return {c, -s, s, c};
00134     }
00135 };

```

## 8.57 include/utils/graph\_drawer.h File Reference

```
#include <string>
#include <stdio.h>
#include <vector>
#include <cmath>
#include "vex.h"
#include "../core/include/utils/geometry.h"
#include "../core/include/utils/vector2d.h"
Include dependency graph for graph_drawer.h:
```



### Classes

- class [GraphDrawer](#)

## 8.58 graph\_drawer.h

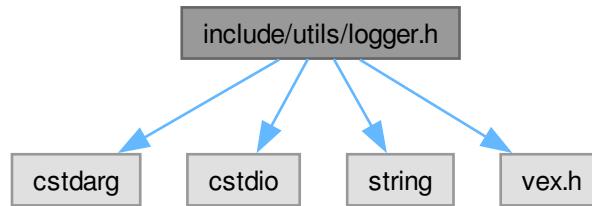
[Go to the documentation of this file.](#)

```

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 };
  
```

## 8.59 include/utils/logger.h File Reference

```
#include <cstdarg>
#include <stdio>
#include <string>
#include "vex.h"
Include dependency graph for logger.h:
```



### Classes

- class [Logger](#)  
*Class to simplify writing to files.*

### Enumerations

- enum [LogLevel](#) {
 DEBUG , NOTICE , WARNING , ERROR ,
 CRITICAL , TIME }
   
*possible values for log filtering*

#### 8.59.1 Enumeration Type Documentation

##### 8.59.1.1 LogLevel

enum [LogLevel](#)

possible values for log filtering

###### Enumerator

DEBUG	
NOTICE	
WARNING	
ERROR	
CRITICAL	
TIME	

## 8.60 logger.h

[Go to the documentation of this file.](#)

```

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
00039     void Log(const std::string &s);
00042
00043     void Log(LogLevel level, const std::string &s);
00044
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 };

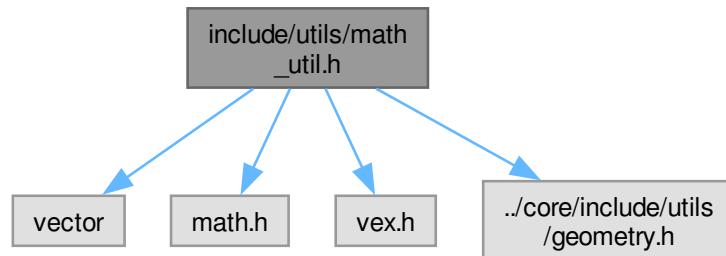
```

## 8.61 include/utils/math\_util.h File Reference

```

#include <vector>
#include "math.h"
#include "vex.h"
#include "../core/include/utils/geometry.h"
Include dependency graph for math_util.h:

```



## Functions

- double [clamp](#) (double value, double low, double high)
- double [lerp](#) (double a, double b, double t)
 

*Linearly intERPolate between values.*
- double [sign](#) (double x)
- double [wrap\\_angle\\_deg](#) (double input)
- double [wrap\\_angle\\_rad](#) (double input)
- double [variance](#) (std::vector< double > const &values, double [mean](#))
- double [mean](#) (std::vector< double > const &values)
- double [covariance](#) (std::vector< std::pair< double, double > > const &points, double meanx, double meany)
- std::pair< double, double > [calculate\\_linear\\_regression](#) (std::vector< std::pair< double, double > > const &points)
- double [estimate\\_path\\_length](#) (const std::vector< [point\\_t](#) > &points)

### 8.61.1 Function Documentation

#### 8.61.1.1 [calculate\\_linear\\_regression\(\)](#)

```
std::pair< double, double > calculate_linear_regression (
    std::vector< std::pair< double, double > > const & points )
```

#### 8.61.1.2 [clamp\(\)](#)

```
double clamp (
    double val,
    double low,
    double high )
```

Constrain the input between a minimum and a maximum value

##### Parameters

<i>val</i>	the value to be restrained
<i>low</i>	the minimum value that will be returned
<i>high</i>	the maximum value that will be returned

#### 8.61.1.3 [covariance\(\)](#)

```
double covariance (
    std::vector< std::pair< double, double > > const & points,
    double meanx,
    double meany )
```

#### 8.61.1.4 [estimate\\_path\\_length\(\)](#)

```
double estimate_path_length (
    const std::vector< point\_t > & points )
```

### 8.61.1.5 lerp()

```
double lerp (
    double a,
    double b,
    double t )
```

Linearly intEРPolate between values.

#### Parameters

<i>a</i>	at $t = 0$ , output = a
<i>b</i>	at $t = 1$ , output = b

#### Returns

a linear mixing of a and b according to t

### 8.61.1.6 mean()

```
double mean (
    std::vector< double > const & values )
```

### 8.61.1.7 sign()

```
double sign (
    double x )
```

Returns the sign of a number

#### Parameters

<i>x</i>	
----------	--

returns the sign +/-1 of x. 0 if x is 0

Returns the sign of a number

#### Parameters

<i>x</i>	
----------	--

returns the sign +/-1 of x. special case at 0 it returns +1

### 8.61.1.8 variance()

```
double variance (
    std::vector< double > const & values,
    double mean )
```

### 8.61.1.9 wrap\_angle\_deg()

```
double wrap_angle_deg (
    double input )
```

### 8.61.1.10 wrap\_angle\_rad()

```
double wrap_angle_rad (
    double input )
```

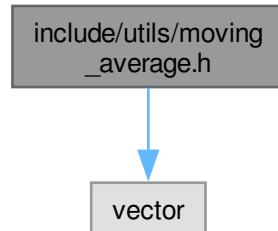
## 8.62 math\_util.h

[Go to the documentation of this file.](#)

```
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);
```

## 8.63 include/utils/moving\_average.h File Reference

```
#include <vector>
Include dependency graph for moving_average.h:
```



### Classes

- class [Filter](#)
- class [MovingAverage](#)
- class [ExponentialMovingAverage](#)

## 8.64 moving\_average.h

[Go to the documentation of this file.](#)

```

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     */
00035     MovingAverage(int buffer_size);
00036     /*
00037     * Create a moving average calculator with a specified default value
00038     * @param buffer_size      The size of the buffer. The number of samples that constitute a valid
00039     * reading
00039     * @param starting_value   The value that the average will be before any data is added
00040     */
00041     MovingAverage(int buffer_size, double starting_value);
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 };

```

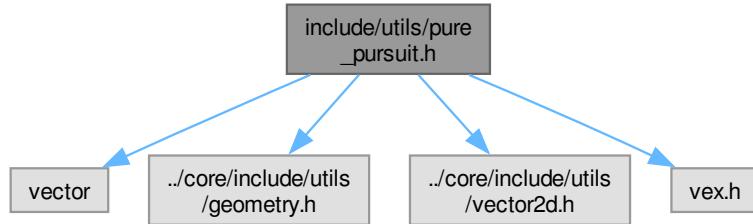
## 8.65 include/utils/pure\_pursuit.h File Reference

```

#include <vector>
#include "../core/include/utils/geometry.h"
#include "../core/include/utils/vector2d.h"
#include "vex.h"

```

Include dependency graph for pure\_pursuit.h:



## Classes

- class [PurePursuit::Path](#)
- struct [PurePursuit::spline](#)
- struct [PurePursuit::hermite\\_point](#)

## Namespaces

- namespace [PurePursuit](#)

## Functions

- std::vector< [point\\_t](#) > PurePursuit::line\_circle\_intersections ([point\\_t](#) center, double r, [point\\_t](#) point1, [point\\_t](#) point2)
- [point\\_t](#) PurePursuit::get\_lookahead (const std::vector< [point\\_t](#) > &path, [pose\\_t](#) robot\_loc, double radius)
- std::vector< [point\\_t](#) > PurePursuit::inject\_path (const std::vector< [point\\_t](#) > &path, double spacing)
- std::vector< [point\\_t](#) > PurePursuit::smooth\_path (const std::vector< [point\\_t](#) > &path, double weight\_data, double weight\_smooth, double tolerance)
- std::vector< [point\\_t](#) > PurePursuit::smooth\_path\_cubic (const std::vector< [point\\_t](#) > &path, double res)
- std::vector< [point\\_t](#) > PurePursuit::smooth\_path\_hermite (const std::vector< [hermite\\_point](#) > &path, double step)
- double PurePursuit::estimate\_remaining\_dist (const std::vector< [point\\_t](#) > &path, [pose\\_t](#) robot\_pose, double radius)

## 8.66 pure\_pursuit.h

[Go to the documentation of this file.](#)

```

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 {
00014     class Path
00015     {
  
```

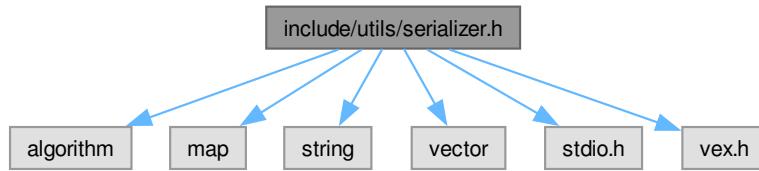
```

00016     public:
00017         Path(std::vector<point_t> points, double radius);
00018
00019         std::vector<point_t> get_points();
00020
00021         double get_radius();
00022
00023         bool is_valid();
00024
00025     private:
00026         std::vector<point_t> points;
00027         double radius;
00028         bool valid;
00029     };
00030     struct spline
00031     {
00032         double a, b, c, d, x_start, x_end;
00033
00034         double getY(double x) {
00035             return a * pow((x - x_start), 3) + b * pow((x - x_start), 2) + c * (x - x_start) + d;
00036         }
00037     };
00038     struct hermite_point
00039     {
00040         double x;
00041         double y;
00042         double dir;
00043         double mag;
00044
00045         point_t getPoint() const {
00046             return {x, y};
00047         }
00048
00049         Vector2D getTangent() const {
00050             return Vector2D(dir, mag);
00051         }
00052     };
00053
00054     extern std::vector<point_t> line_circle_intersections(point_t center, double r, point_t point1,
00055     point_t point2);
00056     extern point_t get_lookahead(const std::vector<point_t> &path, pose_t robot_loc, double radius);
00057
00058     extern std::vector<point_t> inject_path(const std::vector<point_t> &path, double spacing);
00059
00060     extern std::vector<point_t> smooth_path(const std::vector<point_t> &path, double weight_data, double
00061     weight_smooth, double tolerance);
00062
00063     extern std::vector<point_t> smooth_path_cubic(const std::vector<point_t> &path, double res);
00064
00065     extern std::vector<point_t> smooth_path_hermite(const std::vector<hermite_point> &path, double
00066     step);
00067
00068     extern double estimate_remaining_dist(const std::vector<point_t> &path, pose_t robot_pose, double
00069     radius);
00070
00071 }
00072
00073 }
```

## 8.67 include/utils/serializer.h File Reference

```
#include <algorithm>
#include <map>
#include <string>
#include <vector>
#include <stdio.h>
#include <vex.h>
```

Include dependency graph for serializer.h:



## Classes

- class [Serializer](#)  
*Serializes Arbitrary data to a file on the SD Card.*

## Variables

- const char [serialization\\_separator](#) = '\$'  
*character that will be used to separate values*
- const std::size\_t [MAX\\_FILE\\_SIZE](#) = 4096  
*max file size that the system can deal with*

### 8.67.1 Variable Documentation

#### 8.67.1.1 MAX\_FILE\_SIZE

```
const std::size_t MAX_FILE_SIZE = 4096
```

max file size that the system can deal with

#### 8.67.1.2 serialization\_separator

```
const char serialization_separator = '$'
```

character that will be used to separate values

## 8.68 serializer.h

[Go to the documentation of this file.](#)

```

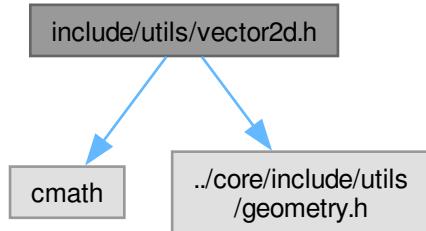
00001 #pragma once
00002 #include <algorithm>
00003 #include <map>
00004 #include <string>
00005 #include <vector>
00006 #include <stdio.h>
00007 #include <vex.h>
00008
00010 const char serialization_separator = '$';
00012 const std::size_t MAX_FILE_SIZE = 4096;
00013
00015 class Serializer
00016 {
00017     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 };

```

## 8.69 include/utils/vector2d.h File Reference

```
#include <cmath>
#include "../core/include/utils/geometry.h"
```

Include dependency graph for vector2d.h:



## Classes

- class [Vector2D](#)

## Macros

- #define [PI](#) 3.141592654

## Functions

- double [deg2rad](#) (double deg)
- double [rad2deg](#) (double r)

### 8.69.1 Macro Definition Documentation

#### 8.69.1.1 PI

```
#define PI 3.141592654
```

### 8.69.2 Function Documentation

#### 8.69.2.1 deg2rad()

```
double deg2rad (
    double deg )
```

General function for converting degrees to radians

##### Parameters

<i>deg</i>	the angle in degrees
------------	----------------------

**Returns**

the angle in radians

General function for converting degrees to radians

**8.69.2.2 rad2deg()**

```
double rad2deg (
    double rad )
```

General function for converting radians to degrees

**Parameters**

<i>r</i>	the angle in radians
----------	----------------------

**Returns**

the angle in degrees

General function for converting radians to degrees

**8.70 vector2d.h**

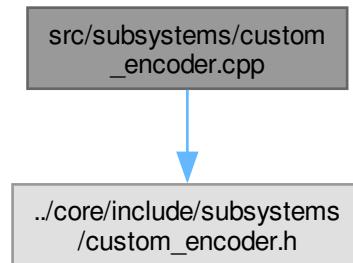
[Go to the documentation of this file.](#)

```
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);
```

## 8.71 README.md File Reference

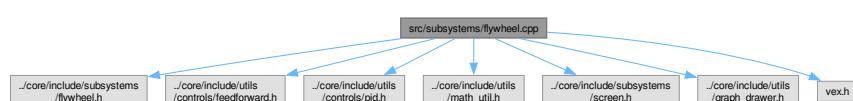
### 8.72 src/subsystems/custom\_encoder.cpp File Reference

```
#include "../core/include/subsystems/custom_encoder.h"
Include dependency graph for custom_encoder.cpp:
```



### 8.73 src/subsystems/flywheel.cpp File Reference

```
#include "../core/include/subsystems/flywheel.h"
#include "../core/include/utils/controls/feedforward.h"
#include "../core/include/utils/controls/pid.h"
#include "../core/include/utils/math_util.h"
#include "../core/include/subsystems/screen.h"
#include "../core/include/utils/graph_drawer.h"
#include "vex.h"
Include dependency graph for flywheel.cpp:
```



#### Classes

- class [FlywheelPage](#)

#### Functions

- int [spinRPMTask](#) (void \*wheelPointer)

### 8.73.1 Function Documentation

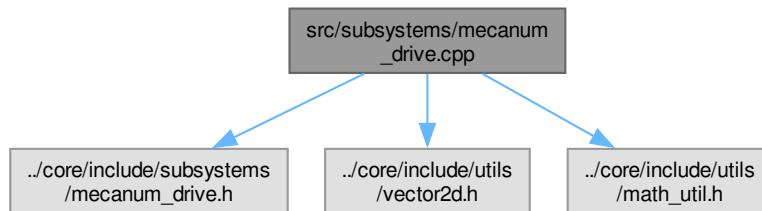
#### 8.73.1.1 spinRPMTask()

```
int spinRPMTask (
    void * wheelPointer )
```

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

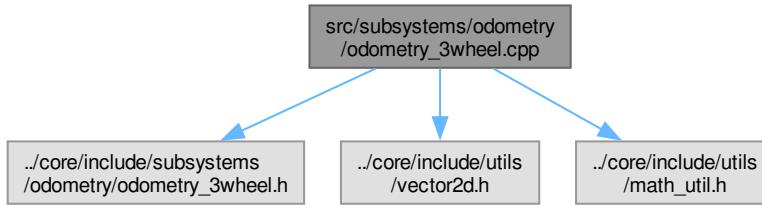
### 8.74 src/subsystems/mecanum\_drive.cpp File Reference

```
#include "../core/include/subsystems/mecanum_drive.h"
#include "../core/include/utils/vector2d.h"
#include "../core/include/utils/math_util.h"
Include dependency graph for mecanum_drive.cpp:
```



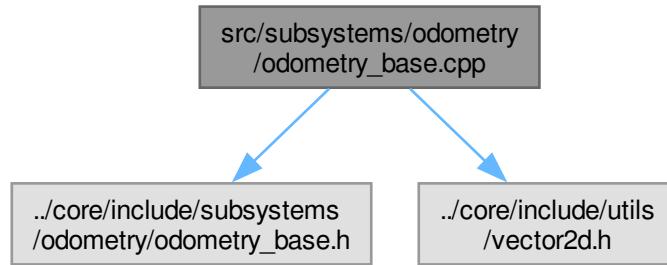
### 8.75 src/subsystems/odometry/odometry\_3wheel.cpp File Reference

```
#include "../core/include/subsystems/odometry/odometry_3wheel.h"
#include "../core/include/utils/vector2d.h"
#include "../core/include/utils/math_util.h"
Include dependency graph for odometry_3wheel.cpp:
```



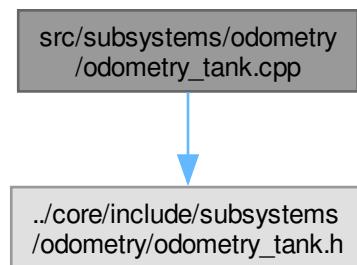
## 8.76 src/subsystems/odometry/odometry\_base.cpp File Reference

```
#include "../core/include/subsystems/odometry/odometry_base.h"
#include "../core/include/utils/vector2d.h"
Include dependency graph for odometry_base.cpp:
```



## 8.77 src/subsystems/odometry/odometry\_tank.cpp File Reference

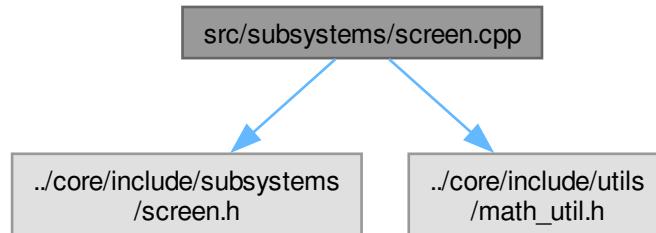
```
#include "../core/include/subsystems/odometry/odometry_tank.h"
Include dependency graph for odometry_tank.cpp:
```



## 8.78 src/subsystems/screen.cpp File Reference

```
#include "../core/include/subsystems/screen.h"
#include "../core/include/utils/math_util.h"
```

Include dependency graph for screen.cpp:



## Classes

- struct `screen::ScreenData`

*The ScreenData class holds the data that will be passed to the screen thread you probably shouldnt have to use it.*

## Namespaces

- namespace `screen`

## Functions

- void `screen::draw_label` (vex::brain::lcd &scr, std::string lbl, `ScreenRect` rect)
- void `screen::draw_widget` (vex::brain::lcd &scr, `WidgetConfig` &widget, `ScreenRect` rect)
- void `screen::start_screen` (vex::brain::lcd &screen, std::vector< `Page` \* > pages, int first\_page=0)
 

*Start the screen background task. Once you start this, no need to draw to the screen manually elsewhere.*
- void `screen::stop_screen` ()
 

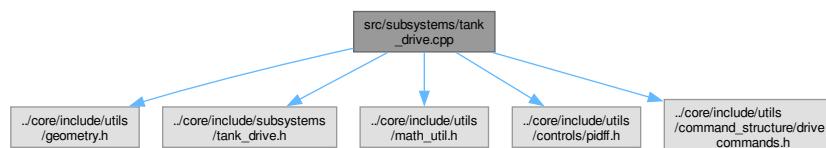
*stops the screen. If you have a drive team that hates fun call this at the start of opcontrol*
- void `screen::prev_page` ()
- void `screen::next_page` ()
- int `screen::in_to_px` (double in)

## 8.79 src/subsystems/tank\_drive.cpp File Reference

```

#include "../core/include/utils/geometry.h"
#include "../core/include/subsystems/tank_drive.h"
#include "../core/include/utils/math_util.h"
#include "../core/include/utils/controls/pidff.h"
#include "../core/include/utils/command_structure/drive_commands.h"
  
```

Include dependency graph for tank\_drive.cpp:



## Variables

- bool `captured_position` = false
- bool `was_breaking` = false

### 8.79.1 Variable Documentation

#### 8.79.1.1 `captured_position`

```
bool captured_position = false
```

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

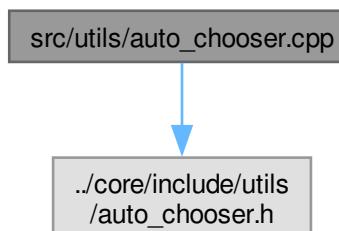
#### 8.79.1.2 `was_breaking`

```
bool was_breaking = false
```

## 8.80 src/utils/auto\_chooser.cpp File Reference

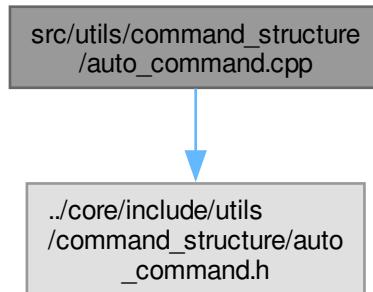
```
#include "../core/include/utils/auto_chooser.h"
```

Include dependency graph for `auto_chooser.cpp`:



## 8.81 src/utils/command\_structure/auto\_command.cpp File Reference

```
#include "../core/include/utils/command_structure/auto_command.h"  
Include dependency graph for auto_command.cpp:
```

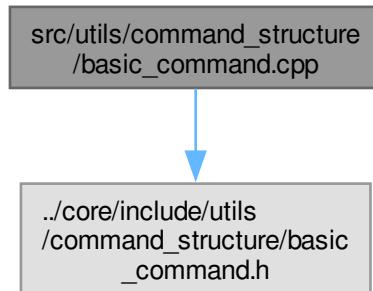


### Classes

- class [OrCondition](#)
- class [AndCondition](#)
- struct [parallel\\_runner\\_info](#)

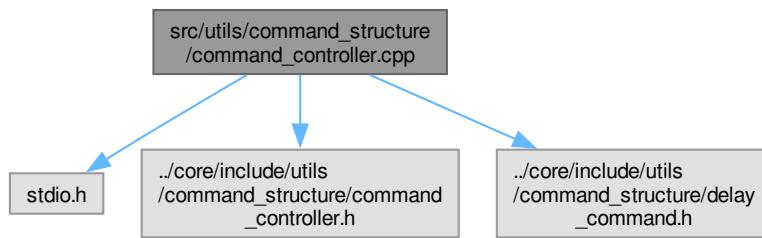
## 8.82 src/utils/command\_structure/basic\_command.cpp File Reference

```
#include "../core/include/utils/command_structure/basic_command.h"  
Include dependency graph for basic_command.cpp:
```



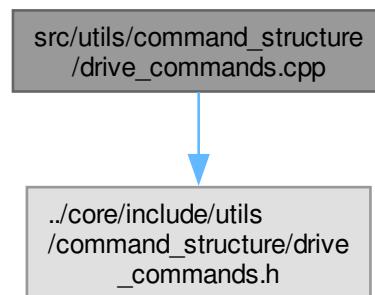
## 8.83 src/utils/command\_structure/command\_controller.cpp File Reference

```
#include <stdio.h>
#include "../core/include/utils/command_structure/command_controller.h"
#include "../core/include/utils/command_structure/delay_command.h"
Include dependency graph for command_controller.cpp:
```



## 8.84 src/utils/command\_structure/drive\_commands.cpp File Reference

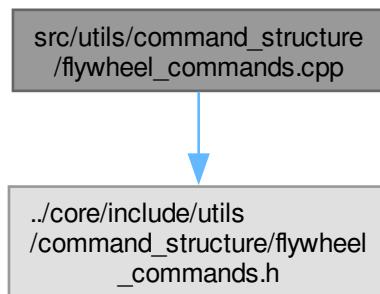
```
#include "../core/include/utils/command_structure/drive_commands.h"
Include dependency graph for drive_commands.cpp:
```



## 8.85 src/utils/command\_structure/flywheel\_commands.cpp File Reference

```
#include "../core/include/utils/command_structure/flywheel_commands.h"
```

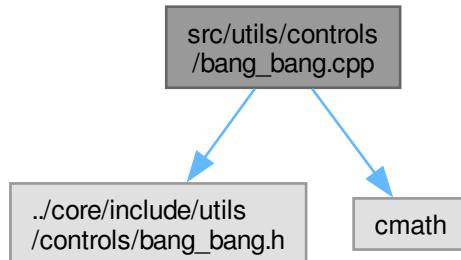
Include dependency graph for flywheel\_commands.cpp:



## 8.86 src/utils/controls/bang\_bang.cpp File Reference

```
#include "../core/include/utils/controls/bang_bang.h"
#include <cmath>
```

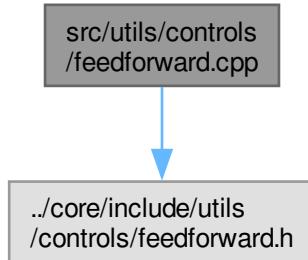
Include dependency graph for bang\_bang.cpp:



## 8.87 src/utils/controls/feedforward.cpp File Reference

```
#include "../core/include/utils/controls/feedforward.h"
```

Include dependency graph for feedforward.cpp:



## Functions

- [FeedForward::ff\\_config\\_t tune\\_feedforward](#) (vex::motor\_group &motor, double pct, double duration)

### 8.87.1 Function Documentation

#### 8.87.1.1 tune\_feedforward()

```
FeedForward::ff_config_t tune_feedforward (
    vex::motor_group & motor,
    double pct,
    double duration )
```

tune\_feedforward takes a group of motors and finds the feedforward config parameters automagically.

##### Parameters

<i>motor</i>	the motor group to use
<i>pct</i>	Maximum velocity in percent (0->1.0)
<i>duration</i>	Amount of time the motors spin for the test

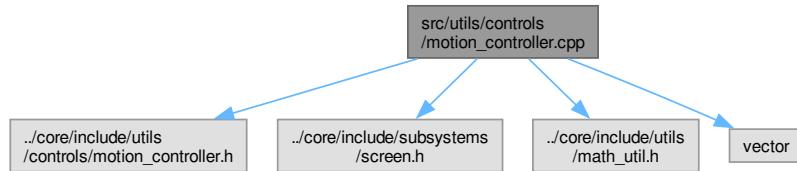
##### Returns

A tuned feedforward object

## 8.88 src/utils/controls/motion\_controller.cpp File Reference

```
#include "../core/include/utils/controls/motion_controller.h"
#include "../core/include/subsystems/screen.h"
#include "../core/include/utils/math_util.h"
```

```
#include <vector>
Include dependency graph for motion_controller.cpp:
```

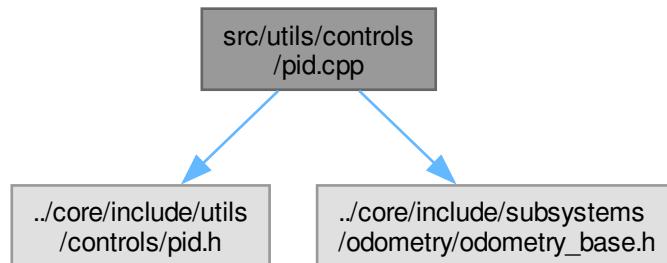


## Classes

- class [MotionControllerPage](#)

## 8.89 src/utils/controls/pid.cpp File Reference

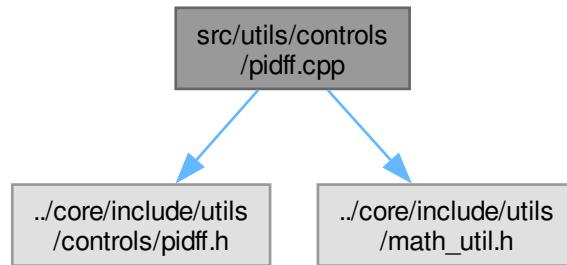
```
#include "../core/include/utils/controls/pid.h"
#include "../core/include/subsystems/odometry/odometry_base.h"
Include dependency graph for pid.cpp:
```



## 8.90 src/utils/controls/pidff.cpp File Reference

```
#include "../core/include/utils/controls/pidff.h"
#include "../core/include/utils/math_util.h"
```

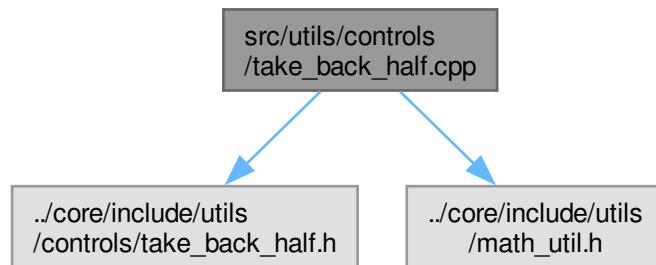
Include dependency graph for pidff.cpp:



## 8.91 src/utils/controls/take\_back\_half.cpp File Reference

```
#include "../core/include/utils/controls/take_back_half.h"
#include "../core/include/utils/math_util.h"
```

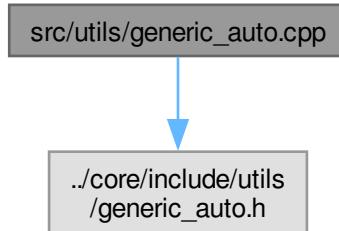
Include dependency graph for take\_back\_half.cpp:



## 8.92 src/utils/generic\_auto.cpp File Reference

```
#include "../core/include/utils/generic_auto.h"
```

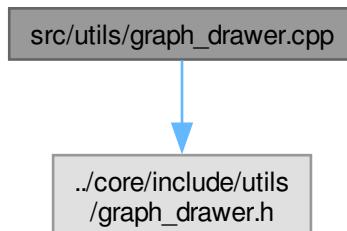
Include dependency graph for generic\_auto.cpp:



## 8.93 src/utils/graph\_drawer.cpp File Reference

```
#include "../core/include/utils/graph_drawer.h"
```

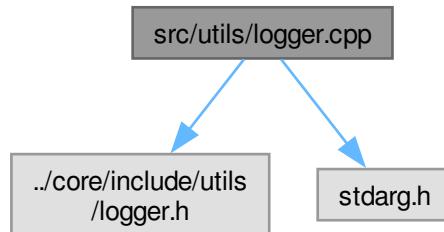
Include dependency graph for graph\_drawer.cpp:



## 8.94 src/utils/logger.cpp File Reference

```
#include "../core/include/utils/logger.h"
#include <stdarg.h>
```

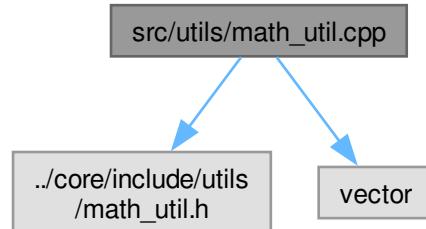
Include dependency graph for logger.cpp:



## 8.95 src/utils/math\_util.cpp File Reference

```
#include "../core/include/utils/math_util.h"
#include <vector>
```

Include dependency graph for math\_util.cpp:



### Macros

- #define PI 3.141592654

### Functions

- double **clamp** (double val, double low, double high)
- double **lerp** (double a, double b, double t)  
*Linearly intERPolate between values.*
- double **sign** (double x)
- double **wrap\_angle\_deg** (double input)
- double **wrap\_angle\_rad** (double input)
- double **mean** (std::vector< double > const &values)

- double `variance` (std::vector< double > const &values, double `mean`)
- double `covariance` (std::vector< std::pair< double, double > > const &points, double `meanx`, double `meany`)
- std::pair< double, double > `calculate_linear_regression` (std::vector< std::pair< double, double > > const &points)
- double `estimate_path_length` (const std::vector< `point_t` > &points)

## 8.95.1 Macro Definition Documentation

### 8.95.1.1 PI

```
#define PI 3.141592654
```

## 8.95.2 Function Documentation

### 8.95.2.1 calculate\_linear\_regression()

```
std::pair< double, double > calculate_linear_regression (
    std::vector< std::pair< double, double > > const & points )
```

### 8.95.2.2 clamp()

```
double clamp (
    double val,
    double low,
    double high )
```

Constrain the input between a minimum and a maximum value

#### Parameters

<i>val</i>	the value to be restrained
<i>low</i>	the minimum value that will be returned
<i>high</i>	the maximum value that will be returned

### 8.95.2.3 covariance()

```
double covariance (
    std::vector< std::pair< double, double > > const & points,
    double meanx,
    double meany )
```

### 8.95.2.4 estimate\_path\_length()

```
double estimate_path_length (
    const std::vector< point_t > & points )
```

### 8.95.2.5 lerp()

```
double lerp (
    double a,
    double b,
    double t )
```

Linearly intERPolate between values.

#### Parameters

<i>a</i>	at $t = 0$ , output = <i>a</i>
<i>b</i>	at $t = 1$ , output = <i>b</i>

#### Returns

a linear mixing of *a* and *b* according to *t*

### 8.95.2.6 mean()

```
double mean (
    std::vector< double > const & values )
```

### 8.95.2.7 sign()

```
double sign (
    double x )
```

Returns the sign of a number

#### Parameters

<i>x</i>	
----------	--

returns the sign +/-1 of *x*. special case at 0 it returns +1

### 8.95.2.8 variance()

```
double variance (
    std::vector< double > const & values,
    double mean )
```

### 8.95.2.9 wrap\_angle\_deg()

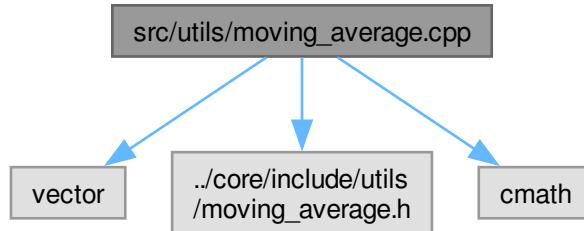
```
double wrap_angle_deg (
    double input )
```

### 8.95.2.10 wrap\_angle\_rad()

```
double wrap_angle_rad (
    double input )
```

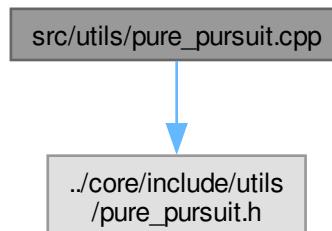
## 8.96 src/utils/moving\_average.cpp File Reference

```
#include <vector>
#include "../core/include/utils/moving_average.h"
#include <cmath>
Include dependency graph for moving_average.cpp:
```



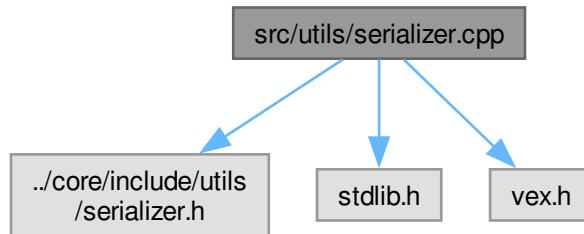
## 8.97 src/utils/pure\_pursuit.cpp File Reference

```
#include "../core/include/utils/pure_pursuit.h"
Include dependency graph for pure_pursuit.cpp:
```



## 8.98 src/utils/serializer.cpp File Reference

```
#include "../core/include/utils/serializer.h"
#include "stdlib.h"
#include "vex.h"
Include dependency graph for serializer.cpp:
```



### Functions

- template<typename T >  
std::vector< char > **to\_bytes** (T value)  
*Convert type to bytes. Overload this for non integer types.*
- template<> std::vector< char > **to\_bytes< std::string >** (std::string str)
- template<typename T >  
T **from\_bytes** (std::vector< char >::const\_iterator &position)  
*Convert bytes to a type.*
- template<> std::string **from\_bytes** (std::vector< char >::const\_iterator &position)
- std::string **sanitize\_name** (std::string s)  
*Replaces funny characters in names so they don't mess with serialization specifiers.*

### 8.98.1 Function Documentation

#### 8.98.1.1 **from\_bytes()** [1/2]

```
template<typename T >
T from_bytes (
    std::vector< char >::const_iterator & position )
```

Convert bytes to a type.

##### Parameters

<i>gets</i>	data from arbitrary bytes. Overload this for non integer types
-------------	--

### 8.98.1.2 `from_bytes()` [2/2]

```
template<>
std::string from_bytes (
    std::vector< char >::const_iterator & position )
```

### 8.98.1.3 `sanitize_name()`

```
std::string sanitize_name (
    std::string s )
```

Replaces funny characters in names so they don't mess with serialization specifiers.

### 8.98.1.4 `to_bytes()`

```
template<typename T >
std::vector< char > to_bytes (
    T value )
```

Convert type to bytes. Overload this for non integer types.

#### Parameters

<code>value</code>	value to convert
--------------------	------------------

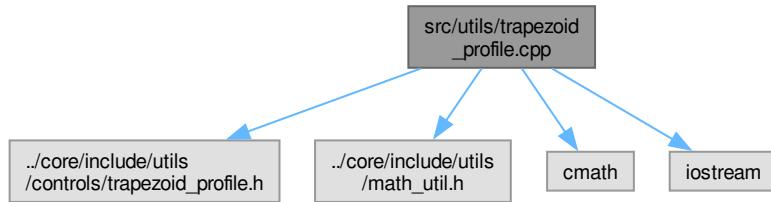
### 8.98.1.5 `to_bytes< std::string >()`

```
template<>
std::vector< char > to_bytes< std::string > (
    std::string str )
```

## 8.99 `src/utils/trapezoid_profile.cpp` File Reference

```
#include "../core/include/utils/controls/trapezoid_profile.h"
#include "../core/include/utils/math_util.h"
#include <cmath>
#include <iostream>
```

Include dependency graph for trapezoid\_profile.cpp:



## Functions

- double `calc_pos` (double t, double a, double v, double si)
- double `calc_vel` (double t, double a, double vi)

## Variables

- const double `EPSILON` = 0.000005

### 8.99.1 Function Documentation

#### 8.99.1.1 calc\_pos()

```
double calc_pos (
    double t,
    double a,
    double v,
    double si ) [inline]
```

#### 8.99.1.2 calc\_vel()

```
double calc_vel (
    double t,
    double a,
    double vi ) [inline]
```

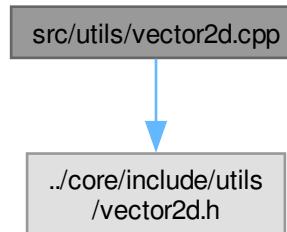
### 8.99.2 Variable Documentation

#### 8.99.2.1 EPSILON

```
const double EPSILON = 0.000005
```

## 8.100 src/utils/vector2d.cpp File Reference

```
#include "../core/include/utils/vector2d.h"  
Include dependency graph for vector2d.cpp:
```



### Functions

- double [deg2rad](#) (double deg)
- double [rad2deg](#) (double rad)

#### 8.100.1 Function Documentation

##### 8.100.1.1 [deg2rad\(\)](#)

```
double deg2rad (  
    double deg )
```

General function for converting degrees to radians

##### 8.100.1.2 [rad2deg\(\)](#)

```
double rad2deg (  
    double rad )
```

General function for converting radians to degrees

# Index

~Branch  
    Branch, 43

~Serializer  
    Serializer, 183

a

    PurePursuit::spline, 193

accel

    motion\_t, 117

    MotionController::m\_profile\_cfg\_t, 111

    OdometryBase, 137

    trapezoid\_profile\_segment\_t, 216

add

    CommandController, 48

    GenericAuto, 92

add\_async

    GenericAuto, 92

add\_cancel\_func

    CommandController, 50

add\_delay

    CommandController, 50

    GenericAuto, 92

add\_entry

    ExponentialMovingAverage, 67

    Filter, 74

    MovingAverage, 126

add\_samples

    GraphDrawer, 94

And

    Condition, 51

AndCondition, 21

    AndCondition, 22

    test, 22

ang\_accel\_deg

    OdometryBase, 137

ang\_speed\_deg

    OdometryBase, 137

ANGULAR

    PID, 157

Async, 22

    Async, 24

    run, 24

auto\_drive

    MecanumDrive, 113

auto\_turn

    MecanumDrive, 114

AutoChooser, 24

    AutoChooser, 26

choice, 28

draw, 26

get\_choice, 26

height, 28

list, 28

update, 26

width, 28

AutoChooser::entry\_t, 64

    name, 65

    rect, 65

AutoCommand, 29

    default\_timeout, 31

    on\_timeout, 30

    run, 30

    timeout\_seconds, 31

    true\_to\_end, 31

    withCancelCondition, 31

    withTimeout, 31

b

    PurePursuit::spline, 193

background\_task

    OdometryBase, 134

BangBang, 32

    BangBang, 33

    get, 33

    init, 33

    is\_on\_target, 33

    set\_limits, 33

    update, 34

BasicSolenoidSet, 34

    BasicSolenoidSet, 36

    run, 36

BasicSpinCommand, 37

    BasicSpinCommand, 38

    percent, 38

    run, 39

    type, 38

    velocity, 38

    voltage, 38

BasicStopCommand, 39

    BasicStopCommand, 41

    run, 41

bool\_or

    Serializer, 183

BrakeType

    TankDrive, 201

Branch, 42

    ~Branch, 43

    Branch, 43

    on\_timeout, 43

    run, 43

Button  
     screen::WidgetConfig, 234

button  
     screen::WidgetConfig, 234

ButtonWidget  
     screen::ButtonWidget, 45

c  
     PurePursuit::spline, 193

calc\_pos  
     trapezoid\_profile.cpp, 315

calc\_vel  
     trapezoid\_profile.cpp, 315

calculate  
     FeedForward, 72  
     TrapezoidProfile, 218

calculate\_linear\_regression  
     math\_util.cpp, 310  
     math\_util.h, 286

calculate\_time\_based  
     TrapezoidProfile, 218

captured\_position  
     tank\_drive.cpp, 301

center  
     Rect, 175

Checkbox  
     screen::WidgetConfig, 234

checkbox  
     screen::WidgetConfig, 234

choice  
     AutoChooser, 28

clamp  
     math\_util.cpp, 310  
     math\_util.h, 286

cmd  
     parallel\_runner\_info, 153

Col  
     screen::WidgetConfig, 234

CommandController, 47  
     add, 48  
     add\_cancel\_func, 50  
     add\_delay, 50  
     CommandController, 47  
     last\_command\_timed\_out, 50  
     run, 50

Condition, 51  
     And, 51  
     Or, 51  
     test, 52

config  
     PID, 160  
     screen::WidgetConfig, 234

contains  
     Rect, 175

control\_continuous  
     Lift< T >, 102

control\_manual  
     Lift< T >, 102

control\_setpoints

Lift< T >, 102

Core, 1

correction\_pid  
     robot\_specs\_t, 179

covariance  
     math\_util.cpp, 310  
     math\_util.h, 286

CRITICAL  
     logger.h, 284

current\_pos  
     OdometryBase, 137

CustomEncoder, 52  
     CustomEncoder, 53  
     position, 53  
     rotation, 53  
     setPosition, 54  
     setRotation, 54  
     velocity, 54

d  
     PID::pid\_config\_t, 161  
     PurePursuit::spline, 193

deadband  
     PID::pid\_config\_t, 161

DEBUG  
     logger.h, 284

default\_timeout  
     AutoCommand, 31

deg2rad  
     vector2d.cpp, 316  
     vector2d.h, 295

DelayCommand, 55  
     DelayCommand, 56  
     run, 56

dimensions  
     Rect, 175

dir  
     PurePursuit::hermite\_point, 95

dist  
     point\_t, 168

dist\_between\_wheels  
     robot\_specs\_t, 179

double\_or  
     Serializer, 183

down\_speed  
     Lift< T >::lift\_cfg\_t, 106

draw  
     AutoChooser, 26  
     FlywheelPage, 80  
     GraphDrawer, 94  
     MotionControllerPage, 124  
     screen::ButtonWidget, 46  
     screen::FunctionPage, 91  
     screen::OdometryPage, 140  
     screen::Page, 150  
     screen::PIDPage, 167  
     screen::SliderWidget, 189  
     screen::StatsPage, 196  
     screen::WidgetPage, 236

draw\_func\_t  
    screen, 18

draw\_label  
    screen, 18

draw\_widget  
    screen, 18, 19

drive  
    MecanumDrive, 114

drive\_arena  
    TankDrive, 202

drive\_correction\_cutoff  
    robot\_specs\_t, 179

drive\_feedback  
    robot\_specs\_t, 179

drive\_forward  
    TankDrive, 202, 203

drive\_gyro\_pid\_conf  
    MecanumDrive::mecanumdrive\_config\_t, 116

drive\_pid\_conf  
    MecanumDrive::mecanumdrive\_config\_t, 116

drive\_raw  
    MecanumDrive, 115

drive\_tank  
    TankDrive, 204

drive\_tank\_raw  
    TankDrive, 204

drive\_wheel\_diam  
    MecanumDrive::mecanumdrive\_config\_t, 116

DriveForwardCmd  
    TankDrive, 206

DriveForwardCommand, 57  
    DriveForwardCommand, 58  
    on\_timeout, 59  
    run, 59

DriveStopCommand, 60  
    DriveStopCommand, 61  
    on\_timeout, 61  
    run, 61

DriveToPointCmd  
    TankDrive, 206

DriveToPointCommand, 62  
    DriveToPointCommand, 63, 64  
    run, 64

duration  
    trapezoid\_profile\_segment\_t, 216

end\_async  
    OdometryBase, 134

end\_task  
    OdometryBase, 138

EPSILON  
    trapezoid\_profile.cpp, 315

ERROR  
    logger.h, 284

error\_method  
    PID::pid\_config\_t, 161

ERROR\_TYPE

PID, 156

estimate\_path\_length  
    math\_util.cpp, 310  
    math\_util.h, 286

estimate\_remaining\_dist  
    PurePursuit, 15

ExponentialMovingAverage, 66  
    add\_entry, 67  
    ExponentialMovingAverage, 67  
    get\_size, 67  
    get\_value, 68

Feedback, 68  
    get, 69  
    init, 69  
    is\_on\_target, 69  
    set\_limits, 70  
    update, 70

FeedForward, 70  
    calculate, 72  
    FeedForward, 71

feedforward.cpp  
    tune\_feedforward, 305

feedforward.h  
    tune\_feedforward, 272

FeedForward::ff\_config\_t, 72  
    kA, 73  
    kG, 73  
    kS, 73  
    kV, 73

ff\_cfg  
    MotionController::m\_profile\_cfg\_t, 111

Filter, 73  
    add\_entry, 74  
    get\_value, 74

first\_cross\_split  
    TakeBackHalf, 199

Flywheel, 74  
    Flywheel, 75  
    FlywheelPage, 78  
    get\_motors, 75  
    get\_target, 75  
    getRPM, 76  
    is\_on\_target, 76  
    Page, 76  
    spin\_manual, 76  
    spin\_rpm, 77  
    SpinRpmCmd, 77  
    spinRPMTask, 78  
    stop, 77  
    WaitUntilUpToSpeedCmd, 78

flywheel.cpp  
    spinRPMTask, 298

FlywheelPage, 79  
    draw, 80  
    Flywheel, 78  
    FlywheelPage, 79  
    update, 80  
    window\_size, 80

FlywheelStopCommand, 81  
     FlywheelStopCommand, 82  
     run, 82  
 FlywheelStopMotorsCommand, 83  
     FlywheelStopMotorsCommand, 84  
     run, 84  
 FlywheelStopNonTasksCommand, 85  
 from\_bytes  
     serializer.cpp, 313  
 from\_min\_and\_size  
     Rect, 175  
 FromRotationDegrees  
     Mat2, 112  
 FunctionCommand, 86  
     FunctionCommand, 87  
     run, 87  
 FunctionCondition, 88  
     FunctionCondition, 89  
     test, 89  
 FunctionPage  
     screen::FunctionPage, 90  
 generic\_auto.h  
     state\_ptr, 280  
 GenericAuto, 91  
     add, 92  
     add\_async, 92  
     add\_delay, 92  
     run, 92  
 get  
     BangBang, 33  
     Feedback, 69  
     MotionController, 120  
     PID, 157  
     PIDFF, 163  
     TakeBackHalf, 198  
 get\_accel  
     OdometryBase, 134  
     TrapezoidProfile, 219  
 get-angular\_accel\_deg  
     OdometryBase, 134  
 get-angular\_speed\_deg  
     OdometryBase, 134  
 get\_async  
     Lift< T >, 103  
 get\_choice  
     AutoChooser, 26  
 get\_dir  
     Vector2D, 226  
 get\_error  
     PID, 157  
 get\_lookahead  
     PurePursuit, 16  
 get\_mag  
     Vector2D, 226  
 get\_max\_v  
     TrapezoidProfile, 219  
 get\_motion  
     MotionController, 120  
 get\_motors  
     Flywheel, 75  
 get\_movement\_time  
     TrapezoidProfile, 219  
 get\_point  
     pose\_t, 171  
 get\_points  
     PurePursuit::Path, 154  
 get\_position  
     OdometryBase, 135  
 get\_radius  
     PurePursuit::Path, 154  
 get\_sensor\_val  
     PID, 157  
     PIDFF, 163  
 get\_setpoint  
     Lift< T >, 103  
 get\_size  
     ExponentialMovingAverage, 67  
     MovingAverage, 126  
 get\_speed  
     OdometryBase, 135  
 get\_target  
     Flywheel, 75  
     PID, 158  
     PIDFF, 163  
 get\_value  
     ExponentialMovingAverage, 68  
     Filter, 74  
     MovingAverage, 127  
 get\_x  
     Vector2D, 226  
 get\_y  
     Vector2D, 227  
 getPoint  
     PurePursuit::hermite\_point, 95  
 getRPM  
     Flywheel, 76  
 getTangent  
     PurePursuit::hermite\_point, 95  
 getY  
     PurePursuit::spline, 192  
 Graph  
     screen::WidgetConfig, 234  
 graph  
     screen::WidgetConfig, 234  
 GraphDrawer, 93  
     add\_samples, 94  
     draw, 94  
     GraphDrawer, 93  
 handle  
     OdometryBase, 138  
 height  
     AutoChooser, 28  
     Rect, 175  
 high  
     screen::SliderConfig, 188  
 hold

Lift< T >, 103  
home  
    Lift< T >, 103  
  
i  
    PID::pid\_config\_t, 161  
IfTimePassed, 96  
    IfTimePassed, 97  
    test, 97  
in\_to\_px  
    screen, 19  
include/robot\_specs.h, 239  
include/subsystems/custom\_encoder.h, 240  
include/subsystems/flywheel.h, 241  
include/subsystems/layout.h, 242  
include/subsystems/lift.h, 243  
include/subsystems/mecanum\_drive.h, 246, 247  
include/subsystems/odometry/odometry\_3wheel.h, 248  
include/subsystems/odometry/odometry\_base.h, 249,  
    250  
include/subsystems/odometry/odometry\_tank.h, 250,  
    251  
include/subsystems/screen.h, 251, 253  
include/subsystems/tank\_drive.h, 256  
include/utils/auto\_chooser.h, 258  
include/utils/command\_structure/auto\_command.h,  
    259, 260  
include/utils/command\_structure/basic\_command.h,  
    262  
include/utils/command\_structure/command\_controller.h,  
    263, 264  
include/utils/command\_structure/delay\_command.h,  
    265  
include/utils/command\_structure/drive\_commands.h,  
    265, 266  
include/utils/command\_structure/flywheel\_commands.h,  
    268, 269  
include/utils/controls/bang\_bang.h, 270  
include/utils/controls/feedback\_base.h, 270, 271  
include/utils/controls/feedforward.h, 271, 273  
include/utils/controls/motion\_controller.h, 273, 274  
include/utils/controls/pid.h, 275  
include/utils/controls/pidff.h, 276, 277  
include/utils/controls/take\_back\_half.h, 277, 278  
include/utils/controls/trapezoid\_profile.h, 278, 279  
include/utils/generic\_auto.h, 279, 280  
include/utils/geometry.h, 281  
include/utils/graph\_drawer.h, 283  
include/utils/logger.h, 284, 285  
include/utils/math\_util.h, 285, 288  
include/utils/moving\_average.h, 289  
include/utils/pure\_pursuit.h, 290, 291  
include/utils/serializer.h, 292, 294  
include/utils/vector2d.h, 294, 296  
index  
    parallel\_runner\_info, 153  
init  
    BangBang, 33  
    Feedback, 69  
MotionController, 120  
PID, 158  
PIDFF, 163  
TakeBackHalf, 198  
inject\_path  
    PurePursuit, 16  
InOrder, 98  
    InOrder, 99  
    on\_timeout, 100  
    run, 100  
int\_or  
    Serializer, 184  
is\_on\_target  
    BangBang, 33  
    Feedback, 69  
    Flywheel, 76  
    MotionController, 120  
    PID, 158  
    PIDFF, 164  
    TakeBackHalf, 198  
is\_valid  
    PurePursuit::Path, 154  
kA  
    FeedForward::ff\_config\_t, 73  
kG  
    FeedForward::ff\_config\_t, 73  
kS  
    FeedForward::ff\_config\_t, 73  
kV  
    FeedForward::ff\_config\_t, 73  
Label  
    screen::WidgetConfig, 234  
label  
    screen::LabelConfig, 100  
    screen::WidgetConfig, 234  
last\_command\_timed\_out  
    CommandController, 50  
lateral\_wheel\_diam  
    MecanumDrive::mecanumdrive\_config\_t, 117  
lerp  
    math\_util.cpp, 310  
    math\_util.h, 286  
Lift  
    Lift< T >, 101  
Lift< T >, 101  
    control\_continuous, 102  
    control\_manual, 102  
    control\_setpoints, 102  
    get\_async, 103  
    get\_setpoint, 103  
    hold, 103  
    home, 103  
    Lift, 101  
    set\_async, 103  
    set\_position, 104  
    set\_sensor\_function, 104  
    set\_sensor\_reset, 104

set\_setpoint, 104  
**Lift< T >::lift\_cfg\_t**, 105  
 down\_speed, 106  
 lift\_pid\_cfg, 106  
 softstop\_down, 106  
 softstop\_up, 106  
 up\_speed, 106  
**lift\_pid\_cfg**  
 Lift< T >::lift\_cfg\_t, 106  
**line\_circle\_intersections**  
 PurePursuit, 16  
**LINEAR**  
 PID, 157  
**list**  
 AutoChooser, 28  
**Log**  
 Logger, 108  
**Logf**  
 Logger, 108  
**Logger**, 106  
 Log, 108  
 Logf, 108  
 Logger, 107  
 LogIn, 109  
 MAX\_FORMAT\_LEN, 109  
 operator=, 109  
**logger.h**  
 CRITICAL, 284  
 DEBUG, 284  
 ERROR, 284  
 LogLevel, 284  
 NOTICE, 284  
 TIME, 284  
 WARNING, 284  
**LogLevel**  
 logger.h, 284  
**LogIn**  
 Logger, 109  
**low**  
 screen::SliderConfig, 188  
**mag**  
 PurePursuit::hermite\_point, 95  
**Mat2**, 111  
 FromRotationDegrees, 112  
 operator\*, 112  
 X11, 112  
 X12, 112  
 X21, 112  
 X22, 112  
**math\_util.cpp**  
 calculate\_linear\_regression, 310  
 clamp, 310  
 covariance, 310  
 estimate\_path\_length, 310  
 lerp, 310  
 mean, 311  
 PI, 310  
 sign, 311  
 variance, 311  
 wrap\_angle\_deg, 311  
 wrap\_angle\_rad, 311  
**math\_util.h**  
 calculate\_linear\_regression, 286  
 clamp, 286  
 covariance, 286  
 estimate\_path\_length, 286  
 lerp, 286  
 mean, 287  
 sign, 287  
 variance, 287  
 wrap\_angle\_deg, 287  
 wrap\_angle\_rad, 288  
**max**  
 Rect, 175  
 SliderCfg, 187  
**MAX\_FILE\_SIZE**  
 serializer.h, 293  
**MAX\_FORMAT\_LEN**  
 Logger, 109  
**MAX\_TRAPEZOID\_PROFILE\_SEGMENTS**  
 trapezoid\_profile.h, 278  
**max\_v**  
 MotionController::m\_profile\_cfg\_t, 111  
**mean**  
 math\_util.cpp, 311  
 math\_util.h, 287  
**mecanum\_drive.h**  
 PI, 247  
**MecanumDrive**, 112  
 auto\_drive, 113  
 auto\_turn, 114  
 drive, 114  
 drive\_raw, 115  
 MecanumDrive, 113  
**MecanumDrive::mecanumdrive\_config\_t**, 116  
 drive\_gyro\_pid\_conf, 116  
 drive\_pid\_conf, 116  
 drive\_wheel\_diam, 116  
 lateral\_wheel\_diam, 117  
 turn\_pid\_conf, 117  
 wheelbase\_width, 117  
**min**  
 Rect, 175  
 SliderCfg, 187  
**modify\_inputs**  
 TankDrive, 206  
**motion\_t**, 117  
 accel, 117  
 pos, 117  
 vel, 118  
**MotionController**, 118  
 get, 120  
 get\_motion, 120  
 init, 120  
 is\_on\_target, 120  
 MotionController, 119

MotionControllerPage, 122  
Page, 121  
set\_limits, 121  
tune\_feedforward, 121  
update, 122  
MotionController::m\_profile\_cfg\_t, 110  
    accel, 111  
    ff\_cfg, 111  
    max\_v, 111  
    pid\_cfg, 111  
MotionControllerPage, 123  
    draw, 124  
    MotionController, 122  
    MotionControllerPage, 123  
    update, 124  
MovingAverage, 124  
    add\_entry, 126  
    get\_size, 126  
    get\_value, 127  
    MovingAverage, 126  
mut  
    OdometryBase, 138  
  
name  
    AutoChooser::entry\_t, 65  
next\_page  
    screen, 19  
None  
    TankDrive, 201  
normalize  
    Vector2D, 227  
NOTICE  
    logger.h, 284  
  
odom\_gear\_ratio  
    robot\_specs\_t, 180  
odom\_wheel\_diam  
    robot\_specs\_t, 180  
Odometry3Wheel, 127  
    Odometry3Wheel, 129  
    tune, 130  
    update, 130  
Odometry3Wheel::odometry3wheel\_cfg\_t, 131  
    off\_axis\_center\_dist, 131  
    wheel\_diam, 131  
    wheelbase\_dist, 131  
odometry\_base.h  
    PI, 249  
OdometryBase, 132  
    accel, 137  
    ang\_accel\_deg, 137  
    ang\_speed\_deg, 137  
    background\_task, 134  
    current\_pos, 137  
    end\_async, 134  
    end\_task, 138  
    get\_accel, 134  
    get\_angular\_accel\_deg, 134  
    get\_angular\_speed\_deg, 134  
    get\_position, 135  
    get\_speed, 135  
    handle, 138  
    mut, 138  
    OdometryBase, 133  
    pos\_diff, 135  
    rot\_diff, 136  
    set\_position, 136  
    SetPositionCmd, 136  
    smallest\_angle, 136  
    speed, 138  
    update, 137  
    zero\_pos, 138  
OdometryPage  
    screen::OdometryPage, 140  
OdometryTank, 141  
    OdometryTank, 142, 143  
    set\_position, 144  
    update, 144  
OdomSetPosition, 144  
    OdomSetPosition, 146  
    run, 147  
off\_axis\_center\_dist  
    Odometry3Wheel::odometry3wheel\_cfg\_t, 131  
on\_target\_time  
    PID::pid\_config\_t, 161  
on\_timeout  
    AutoCommand, 30  
    Branch, 43  
    DriveForwardCommand, 59  
    DriveStopCommand, 61  
    InOrder, 100  
    Parallel, 152  
    PurePursuitCommand, 173  
    RepeatUntil, 178  
    TurnDegreesCommand, 222  
    TurnToHeadingCommand, 224  
onclick  
    screen::ButtonConfig, 44  
onupdate  
    screen::CheckboxConfig, 47  
operator+  
    point\_t, 169  
    Vector2D, 227  
operator-  
    point\_t, 169  
    Vector2D, 228  
operator/  
    point\_t, 170  
operator=  
    Logger, 109  
operator==  
    point\_t, 170  
operator\*  
    Mat2, 112  
    point\_t, 169  
    Vector2D, 227  
Or

Condition, 51  
 OrCondition, 147  
   OrCondition, 148  
   test, 148

p

PID::pid\_config\_t, 161

Page

Flywheel, 76  
 MotionController, 121

page

screen::ScreenData, 181

pages

screen::ScreenData, 181

Parallel, 150

on\_timeout, 152

Parallel, 152

run, 152

parallel\_runner\_info, 153

cmd, 153  
 index, 153  
 runners, 153

Path

PurePursuit::Path, 154

percent

BasicSpinCommand, 38

PI

math\_util.cpp, 310  
 mecanum\_drive.h, 247  
 odometry\_base.h, 249  
 tank\_drive.h, 256  
 vector2d.h, 295

PID, 155

ANGULAR, 157  
 config, 160  
 ERROR\_TYPE, 156  
 get, 157  
 get\_error, 157  
 get\_sensor\_val, 157  
 get\_target, 158  
 init, 158  
 is\_on\_target, 158  
 LINEAR, 157  
 PID, 157  
 reset, 159  
 set\_limits, 159  
 set\_target, 159  
 update, 159

pid

PIDFF, 166

PID::pid\_config\_t, 160

d, 161  
 deadband, 161  
 error\_method, 161  
 i, 161  
 on\_target\_time, 161  
 p, 161

pid\_cfg

MotionController::m\_profile\_cfg\_t, 111

PIDFF, 162  
   get, 163  
   get\_sensor\_val, 163  
   get\_target, 163  
   init, 163  
   is\_on\_target, 164  
   pid, 166  
   PIDFF, 163  
   reset, 164  
   set\_limits, 164  
   set\_target, 164  
   update, 165

PIDPage

screen::PIDPage, 167

point

Vector2D, 228

point\_t, 168

dist, 168  
 operator+, 169  
 operator-, 169  
 operator/, 170  
 operator==, 170  
 operator\*, 169  
 x, 170  
 y, 170

pos

motion\_t, 117

pos\_after

trapezoid\_profile\_segment\_t, 216

pos\_diff

OdometryBase, 135

pose\_t, 170

get\_point, 171  
 rot, 171  
 x, 171  
 y, 171

position

CustomEncoder, 53

prev\_page

screen, 19

pure\_pursuit

TankDrive, 208, 209

PurePursuit, 15

estimate\_remaining\_dist, 15  
 get\_lookinghead, 16  
 inject\_path, 16  
 line\_circle\_intersections, 16  
 smooth\_path, 16  
 smooth\_path\_cubic, 16  
 smooth\_path\_hermite, 17

PurePursuit::hermite\_point, 95

dir, 95  
 getPoint, 95  
 getTangent, 95  
 mag, 95  
 x, 95  
 y, 96

PurePursuit::Path, 154

get\_points, 154  
get\_radius, 154  
is\_valid, 154  
Path, 154  
PurePursuit::spline, 192  
    a, 193  
    b, 193  
    c, 193  
    d, 193  
    getY, 192  
    x\_end, 193  
    x\_start, 193  
PurePursuitCmd  
    TankDrive, 209, 210  
PurePursuitCommand, 172  
    on\_timeout, 173  
    PurePursuitCommand, 173  
    run, 173  
rad2deg  
    vector2d.cpp, 316  
    vector2d.h, 296  
README.md, 297  
Rect, 174  
    center, 175  
    contains, 175  
    dimensions, 175  
    from\_min\_and\_size, 175  
    height, 175  
    max, 175  
    min, 175  
    width, 175  
rect  
    AutoChooser::entry\_t, 65  
RepeatUntil, 176  
    on\_timeout, 178  
    RepeatUntil, 177  
    run, 178  
reset  
    PID, 159  
    PIDFF, 164  
reset\_auto  
    TankDrive, 210  
robot\_radius  
    robot\_specs\_t, 180  
robot\_specs\_t, 178  
    correction\_pid, 179  
    dist\_between\_wheels, 179  
    drive\_correction\_cutoff, 179  
    drive\_feedback, 179  
    odom\_gear\_ratio, 180  
    odom\_wheel\_diam, 180  
    robot\_radius, 180  
    turn\_feedback, 180  
rot  
    pose\_t, 171  
rot\_diff  
    OdometryBase, 136  
rotation  
    CustomEncoder, 53  
Row  
    screen::WidgetConfig, 234  
run  
    Async, 24  
    AutoCommand, 30  
    BasicSolenoidSet, 36  
    BasicSpinCommand, 39  
    BasicStopCommand, 41  
    Branch, 43  
    CommandController, 50  
    DelayCommand, 56  
    DriveForwardCommand, 59  
    DriveStopCommand, 61  
    DriveToPointCommand, 64  
    FlywheelStopCommand, 82  
    FlywheelStopMotorsCommand, 84  
    FunctionCommand, 87  
    GenericAuto, 92  
    InOrder, 100  
    OdomSetPosition, 147  
    Parallel, 152  
    PurePursuitCommand, 173  
    RepeatUntil, 178  
    SpinRPMCommand, 192  
    TurnDegreesCommand, 222  
    TurnToHeadingCommand, 224  
    WaitUntilCondition, 230  
    WaitUntilUpToSpeedCommand, 232  
runners  
    parallel\_runner\_info, 153  
sanitize\_name  
    serializer.cpp, 314  
save\_to\_disk  
    Serializer, 184  
screen, 17  
    draw\_func\_t, 18  
    draw\_label, 18  
    draw\_widget, 18, 19  
    in\_to\_px, 19  
    next\_page, 19  
    prev\_page, 19  
    screen::ScreenData, 181  
    start\_screen, 19  
    stop\_screen, 20  
    update\_func\_t, 18  
screen::ButtonConfig, 44  
    onclick, 44  
screen::ButtonWidget, 44  
    ButtonWidget, 45  
    draw, 46  
    update, 46  
screen::CheckboxConfig, 46  
    onupdate, 47  
screen::FunctionPage, 89  
    draw, 91  
    FunctionPage, 90  
    update, 91

screen::LabelConfig, 100  
     label, 100  
 screen::OdometryPage, 139  
     draw, 140  
     OdometryPage, 140  
     update, 140  
 screen::Page, 149  
     draw, 150  
     update, 150  
 screen::PIDPage, 166  
     draw, 167  
     PIDPage, 167  
     update, 167  
 screen::ScreenData, 180  
     page, 181  
     pages, 181  
     screen, 181  
     ScreenData, 181  
 screen::ScreenRect, 181  
     x1, 182  
     x2, 182  
     y1, 182  
     y2, 182  
 screen::SizedWidget, 186  
     size, 187  
     widget, 187  
 screen::SliderConfig, 188  
     high, 188  
     low, 188  
     val, 188  
 screen::SliderWidget, 188  
     draw, 189  
     SliderWidget, 189  
     update, 189  
 screen::StatsPage, 194  
     draw, 196  
     StatsPage, 195  
     update, 196  
 screen::TextConfig, 214  
     text, 214  
 screen::WidgetConfig, 233  
     Button, 234  
     button, 234  
     Checkbox, 234  
     checkbox, 234  
     Col, 234  
     config, 234  
     Graph, 234  
     graph, 234  
     Label, 234  
     label, 234  
     Row, 234  
     Slider, 234  
     slider, 234  
     Text, 234  
     text, 234  
     Type, 233  
     type, 234

widgets, 235  
 screen::WidgetPage, 235  
     draw, 236  
     update, 236  
     WidgetPage, 236  
 ScreenData  
     screen::ScreenData, 181  
 serialization\_separator  
     serializer.h, 293  
 Serializer, 182  
     ~Serializer, 183  
     bool\_or, 183  
     double\_or, 183  
     int\_or, 184  
     save\_to\_disk, 184  
     Serializer, 183  
     set\_bool, 184  
     set\_double, 185  
     set\_int, 185  
     set\_string, 185  
     string\_or, 185  
 serializer.cpp  
     from\_bytes, 313  
     sanitize\_name, 314  
     to\_bytes, 314  
     to\_bytes< std::string >, 314  
 serializer.h  
     MAX\_FILE\_SIZE, 293  
     serialization\_separator, 293  
 set\_accel  
     TrapezoidProfile, 219  
 set\_async  
     Lift< T >, 103  
 set\_bool  
     Serializer, 184  
 set\_double  
     Serializer, 185  
 set\_endpts  
     TrapezoidProfile, 219  
 set\_int  
     Serializer, 185  
 set\_limits  
     BangBang, 33  
     Feedback, 70  
     MotionController, 121  
     PID, 159  
     PIDFF, 164  
     TakeBackHalf, 199  
 set\_max\_v  
     TrapezoidProfile, 220  
 set\_position  
     Lift< T >, 104  
     OdometryBase, 136  
     OdometryTank, 144  
 set\_sensor\_function  
     Lift< T >, 104  
 set\_sensor\_reset  
     Lift< T >, 104

set\_setpoint  
    Lift< T >, 104

set\_string  
    Serializer, 185

set\_target  
    PID, 159  
    PIDFF, 164

set\_vel\_endpts  
    TrapezoidProfile, 220

setPosition  
    CustomEncoder, 54

SetPositionCmd  
    OdometryBase, 136

setRotation  
    CustomEncoder, 54

sign  
    math\_util.cpp, 311  
    math\_util.h, 287

size  
    screen::SizedWidget, 187

Slider  
    screen::WidgetConfig, 234

slider  
    screen::WidgetConfig, 234

SliderCfg, 187  
    max, 187  
    min, 187  
    val, 187

SliderWidget  
    screen::SliderWidget, 189

smallest\_angle  
    OdometryBase, 136

Smart  
    TankDrive, 201

smooth\_path  
    PurePursuit, 16

smooth\_path\_cubic  
    PurePursuit, 16

smooth\_path\_hermite  
    PurePursuit, 17

softstop\_down  
    Lift< T >::lift\_cfg\_t, 106

softstop\_up  
    Lift< T >::lift\_cfg\_t, 106

speed  
    OdometryBase, 138

spin\_manual  
    Flywheel, 76

spin\_rpm  
    Flywheel, 77

SpinRpmCmd  
    Flywheel, 77

SpinRPMCommand, 190  
    run, 192  
    SpinRPMCommand, 191

spinRPMTask  
    Flywheel, 78  
    flywheel.cpp, 298

src/subsystems/custom\_encoder.cpp, 297

src/subsystems/flywheel.cpp, 297

src/subsystems/mecanum\_drive.cpp, 298

src/subsystems/odometry/odometry\_3wheel.cpp, 298

src/subsystems/odometry/odometry\_base.cpp, 299

src/subsystems/odometry/odometry\_tank.cpp, 299

src/subsystems/screen.cpp, 299

src/subsystems/tank\_drive.cpp, 300

src/utils/auto\_chooser.cpp, 301

src/utils/command\_structure/auto\_command.cpp, 302

src/utils/command\_structure/basic\_command.cpp, 302

src/utils/command\_structure/command\_controller.cpp, 303

src/utils/command\_structure/drive\_commands.cpp, 303

src/utils/command\_structure/flywheel\_commands.cpp, 303

src/utils/controls/bang\_bang.cpp, 304

src/utils/controls/feedforward.cpp, 304

src/utils/controls/motion\_controller.cpp, 305

src/utils/controls/pid.cpp, 306

src/utils/controls/pidff.cpp, 306

src/utils/controls/take\_back\_half.cpp, 307

src/utils/generic\_auto.cpp, 307

src/utils/graph\_drawer.cpp, 308

src/utils/logger.cpp, 308

src/utils/math\_util.cpp, 309

src/utils/moving\_average.cpp, 312

src/utils/pure\_pursuit.cpp, 312

src/utils/serializer.cpp, 313

src/utils/trapezoid\_profile.cpp, 314

src/utils/vector2d.cpp, 316

start\_screen  
    screen, 19

state\_ptr  
    generic\_auto.h, 280

StatsPage  
    screen::StatsPage, 195

stop  
    Flywheel, 77  
    TankDrive, 210

stop\_screen  
    screen, 20

string\_or  
    Serializer, 185

TakeBackHalf, 197  
    first\_cross\_split, 199  
    get, 198  
    init, 198  
    is\_on\_target, 198  
    set\_limits, 199  
    TakeBackHalf, 198  
    TBH\_gain, 199  
    update, 199

tank\_drive.cpp  
    captured\_position, 301  
    was\_breaking, 301

tank\_drive.h  
    PI, 256

TankDrive, 200  
 BrakeType, 201  
 drive\_arcade, 202  
 drive\_forward, 202, 203  
 drive\_tank, 204  
 drive\_tank\_raw, 204  
 drive\_to\_point, 204, 205  
 DriveForwardCmd, 206  
 DriveToPointCmd, 206  
 modify\_inputs, 206  
 None, 201  
 pure\_pursuit, 208, 209  
 PurePursuitCmd, 209, 210  
 reset\_auto, 210  
 Smart, 201  
 stop, 210  
 TankDrive, 201  
 turn\_degrees, 210, 211  
 turn\_to\_heading, 211, 212  
 TurnDegreesCmd, 213  
 TurnToHeadingCmd, 213  
 ZeroVelocity, 201  
 TBH\_gain  
 TakeBackHalf, 199  
 test  
 AndCondition, 22  
 Condition, 52  
 FunctionCondition, 89  
 IfTimePassed, 97  
 OrCondition, 148  
 TimesTestedCondition, 215  
 Text  
 screen::WidgetConfig, 234  
 text  
 screen::TextConfig, 214  
 screen::WidgetConfig, 234  
 TIME  
 logger.h, 284  
 timeout\_seconds  
 AutoCommand, 31  
 TimesTestedCondition, 214  
 test, 215  
 TimesTestedCondition, 215  
 to\_bytes  
 serializer.cpp, 314  
 to\_bytes< std::string >  
 serializer.cpp, 314  
 trapezoid\_profile.cpp  
 calc\_pos, 315  
 calc\_vel, 315  
 EPSILON, 315  
 trapezoid\_profile.h  
 MAX\_TRAPEZOID\_PROFILE\_SEGMENTS, 278  
 trapezoid\_profile\_segment\_t, 215  
 accel, 216  
 duration, 216  
 pos\_after, 216  
 vel\_after, 216  
 TrapezoidProfile, 216  
 calculate, 218  
 calculate\_time\_based, 218  
 get\_accel, 219  
 get\_max\_v, 219  
 get\_movement\_time, 219  
 set\_accel, 219  
 set\_endpts, 219  
 set\_max\_v, 220  
 set\_vel\_endpts, 220  
 TrapezoidProfile, 218  
 true\_to\_end  
 AutoCommand, 31  
 tune  
 Odometry3Wheel, 130  
 tune\_feedforward  
 feedforward.cpp, 305  
 feedforward.h, 272  
 MotionController, 121  
 turn\_degrees  
 TankDrive, 210, 211  
 turn\_feedback  
 robot\_specs\_t, 180  
 turn\_pid\_conf  
 MecanumDrive::mecanumdrive\_config\_t, 117  
 turn\_to\_heading  
 TankDrive, 211, 212  
 TurnDegreesCmd  
 TankDrive, 213  
 TurnDegreesCommand, 220  
 on\_timeout, 222  
 run, 222  
 TurnDegreesCommand, 222  
 TurnToHeadingCmd  
 TankDrive, 213  
 TurnToHeadingCommand, 223  
 on\_timeout, 224  
 run, 224  
 TurnToHeadingCommand, 224  
 Type  
 screen::WidgetConfig, 233  
 type  
 BasicSpinCommand, 38  
 screen::WidgetConfig, 234  
 up\_speed  
 Lift< T >::lift\_cfg\_t, 106  
 update  
 AutoChooser, 26  
 BangBang, 34  
 Feedback, 70  
 FlywheelPage, 80  
 MotionController, 122  
 MotionControllerPage, 124  
 Odometry3Wheel, 130  
 OdometryBase, 137  
 OdometryTank, 144  
 PID, 159  
 PIDFF, 165

screen::ButtonWidget, 46  
screen::FunctionPage, 91  
screen::OdometryPage, 140  
screen::Page, 150  
screen::PIDPage, 167  
screen::SliderWidget, 189  
screen::StatsPage, 196  
screen::WidgetPage, 236  
TakeBackHalf, 199  
update\_func\_t  
    screen, 18

val  
    screen::SliderConfig, 188  
    SliderCfg, 187

variance  
    math\_util.cpp, 311  
    math\_util.h, 287

Vector2D, 225  
    get\_dir, 226  
    get\_mag, 226  
    get\_x, 226  
    get\_y, 227  
    normalize, 227  
    operator+, 227  
    operator-, 228  
    operator\*, 227  
    point, 228  
    Vector2D, 225, 226

vector2d.cpp  
    deg2rad, 316  
    rad2deg, 316

vector2d.h  
    deg2rad, 295  
    PI, 295  
    rad2deg, 296

vel  
    motion\_t, 118

vel\_after  
    trapezoid\_profile\_segment\_t, 216

velocity  
    CustomEncoder, 54

veocity  
    BasicSpinCommand, 38

voltage  
    BasicSpinCommand, 38

WaitUntilCondition, 229  
    run, 230

    WaitUntilCondition, 230

WaitUntilUpToSpeedCmd  
    Flywheel, 78

WaitUntilUpToSpeedCommand, 231  
    run, 232

    WaitUntilUpToSpeedCommand, 232

WARNING  
    logger.h, 284

was\_breaking  
    tank\_drive.cpp, 301

wheel\_diam  
    Odometry3Wheel::odometry3wheel\_cfg\_t, 131

wheelbase\_dist  
    Odometry3Wheel::odometry3wheel\_cfg\_t, 131

wheelbase\_width  
    MecanumDrive::mecanumdrive\_config\_t, 117

widget  
    screen::SizedWidget, 187

WidgetPage  
    screen::WidgetPage, 236

widgets  
    screen::WidgetConfig, 235

width  
    AutoChooser, 28  
    Rect, 175

window\_size  
    FlywheelPage, 80

withCancelCondition  
    AutoCommand, 31

withTimeout  
    AutoCommand, 31

wrap\_angle\_deg  
    math\_util.cpp, 311  
    math\_util.h, 287

wrap\_angle\_rad  
    math\_util.cpp, 311  
    math\_util.h, 288

x  
    point\_t, 170  
    pose\_t, 171  
    PurePursuit::hermite\_point, 95

x1  
    screen::ScreenRect, 182

X11  
    Mat2, 112

X12  
    Mat2, 112

x2  
    screen::ScreenRect, 182

X21  
    Mat2, 112

X22  
    Mat2, 112

x\_end  
    PurePursuit::spline, 193

x\_start  
    PurePursuit::spline, 193

y  
    point\_t, 170  
    pose\_t, 171  
    PurePursuit::hermite\_point, 96

y1  
    screen::ScreenRect, 182

y2  
    screen::ScreenRect, 182

zero\_pos

OdometryBase, [138](#)  
ZeroVelocity  
  TankDrive, [201](#)