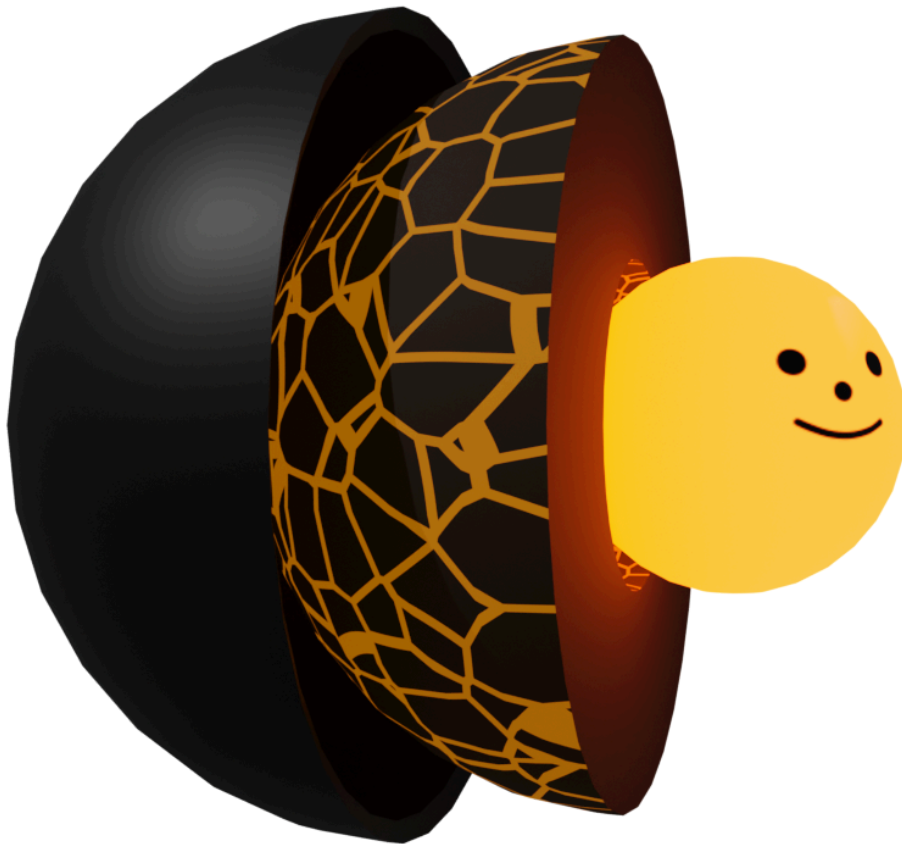


# RIT VEXU Software Engineering Notebook

2023-2024

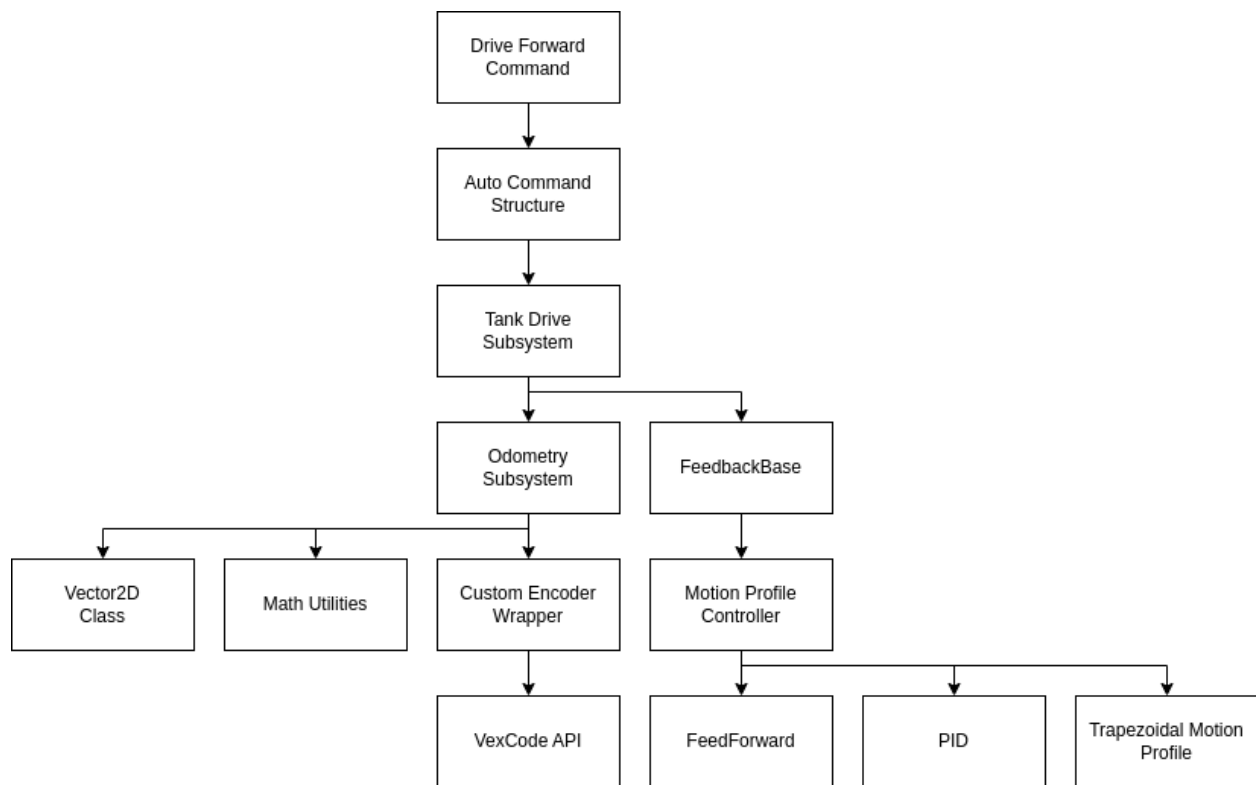


# Overview

## 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.



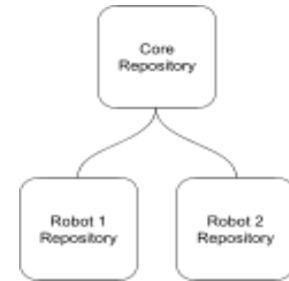
*Figure 1 - Example of Object-Oriented Programming*

## 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*

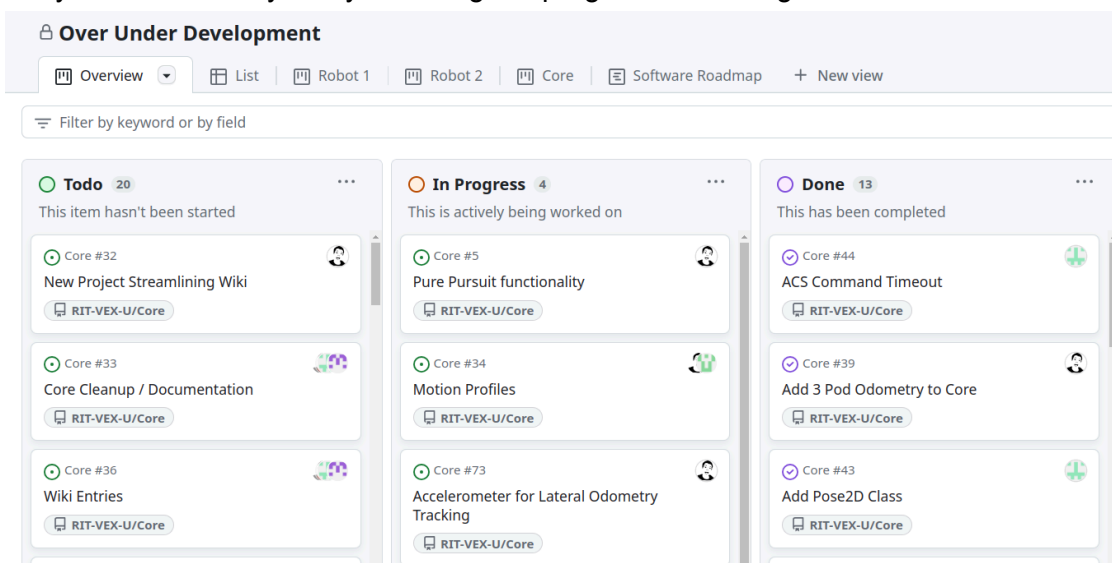
## Git Subrepo

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.



*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.

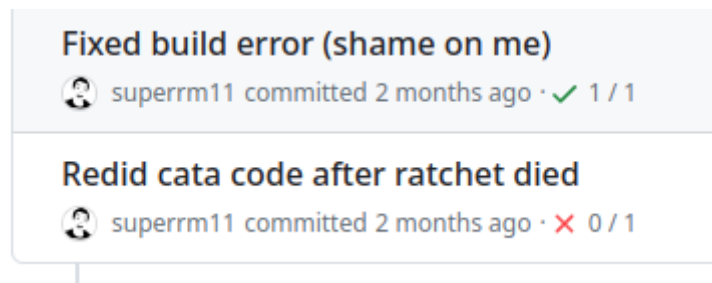


Figure 4: Continuous Integration directly improves the quality of our code.

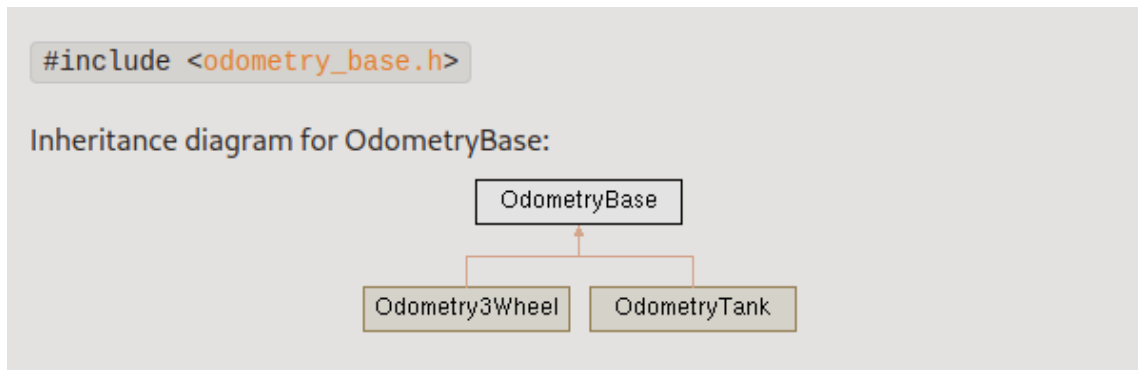


Figure 5: Automatically generated documentation.

## Auto-Notebook

Alongside the automatic 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
  - Tank (same old)
  - ~~- GPS + Odometry~~
- Drivetrain
  - Tank Drive
    - What it do
      - DriveForward, TurnToHeading
      - DriveToPoint
    - Whats new
      - Pure pursuit
      - Brake mode

# Odometry

## Tank Odometry

// todo idk stuff here

## 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.

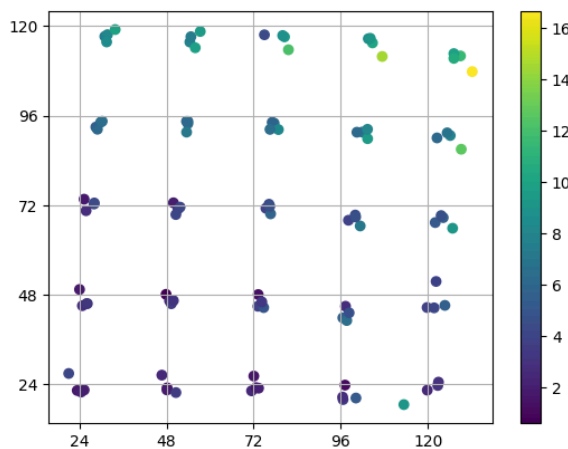


Figure 7 - Raw Data Points

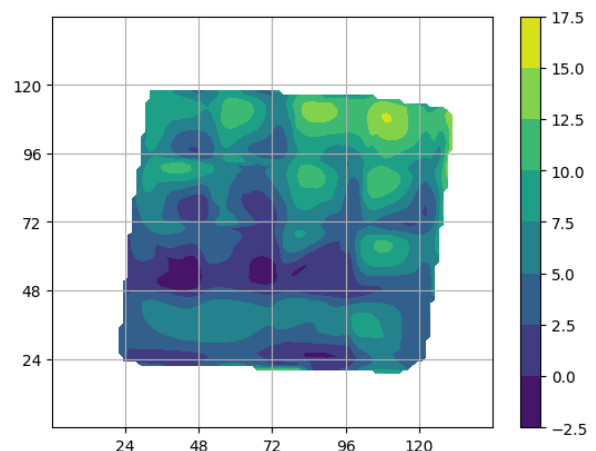


Figure 8 - Error Heat Map

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:

$$\alpha = (|\vec{c} - \vec{p}| \cdot |1|\cos\theta) * \frac{\sqrt{(72-x)^2 + (72-y)^2}}{101.8} * q$$

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

where  $\vec{c}$  is the constant vector of a point in the center of the field (x=72in, y=72in),  $\vec{p}$  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 delta 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};
```

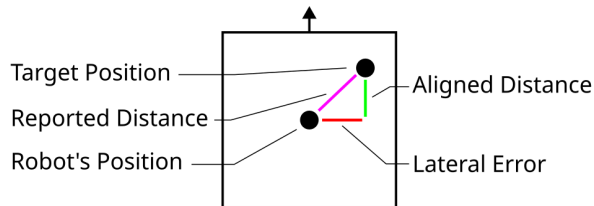
```

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

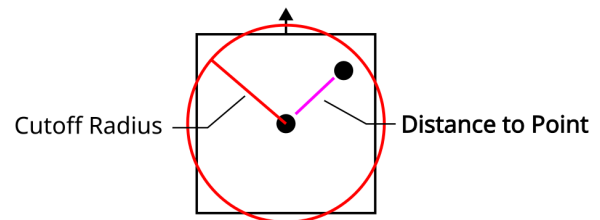
Vector2D point_vec(pos_diff_pt);

```

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 X. 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 X - Distance Modification*



*Figure X - 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.

```

// Logic & math - correction PID, distance cutoffs
// drive_to_point() - header declaration + examples

```

### Pure Pursuit

```

// Logic & math - grab stuff from wiki, funny lookahead circle image
// pure_pursuit() - header declaration + examples

```

# 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.

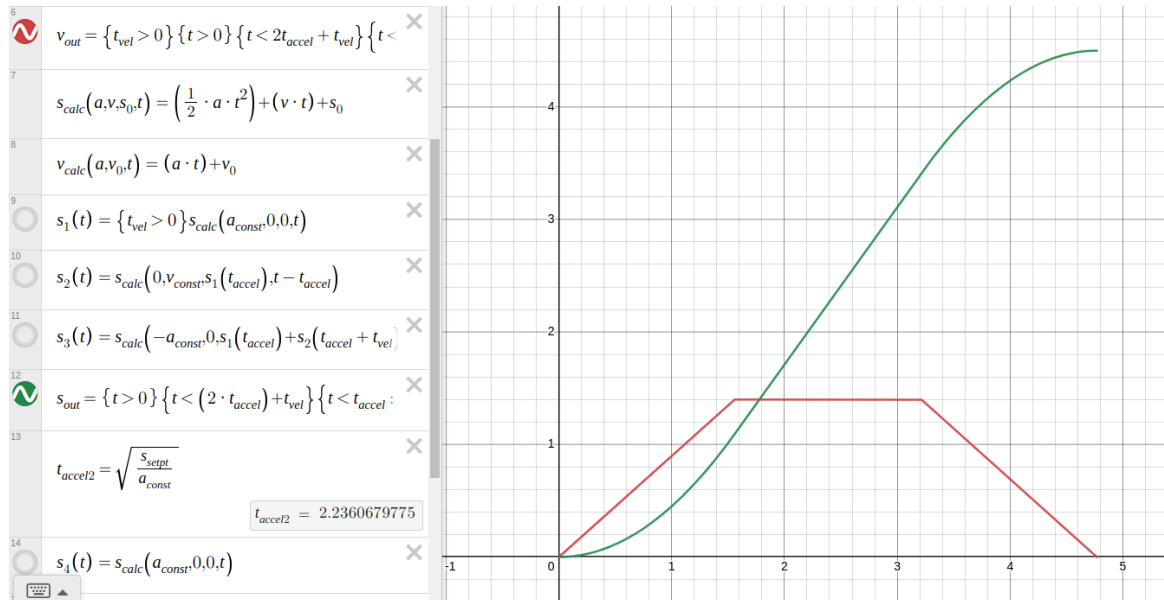
## Generic Feedback

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 and 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 rigidity of a pure feedforward controller by adapting dynamically to changing conditions in competition scenarios.



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;
}

```

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 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))
};

```

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.

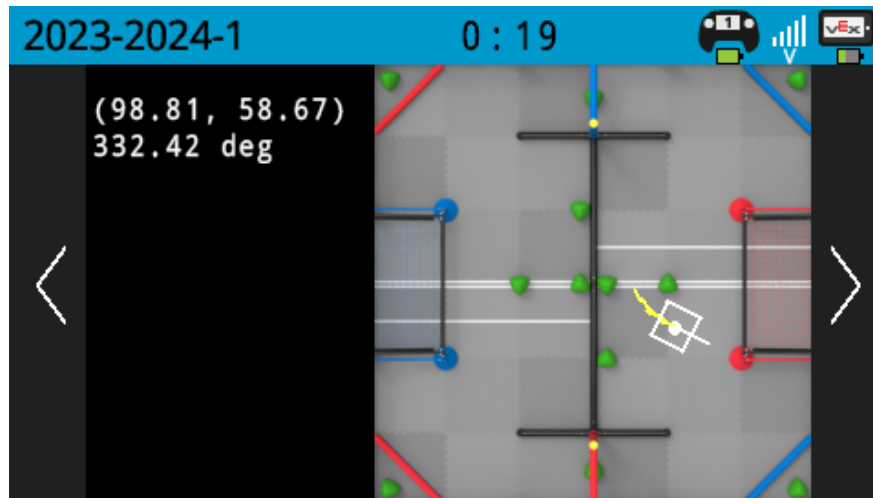
```
pages = {  
    new AutoChooser({"Auto 1", "Auto 2", "Auto 3", "Auto 4"}),  
    new screen::StatsPage(motor_names),  
    new screen::OdometryPage(odom, 12, 12, true),  
    cata_sys.Page(),  
};  
  
screen::start_screen(Brain.Screen, pages);
```

Configuration for the screen subsystem

## Pages

### Odometry Page

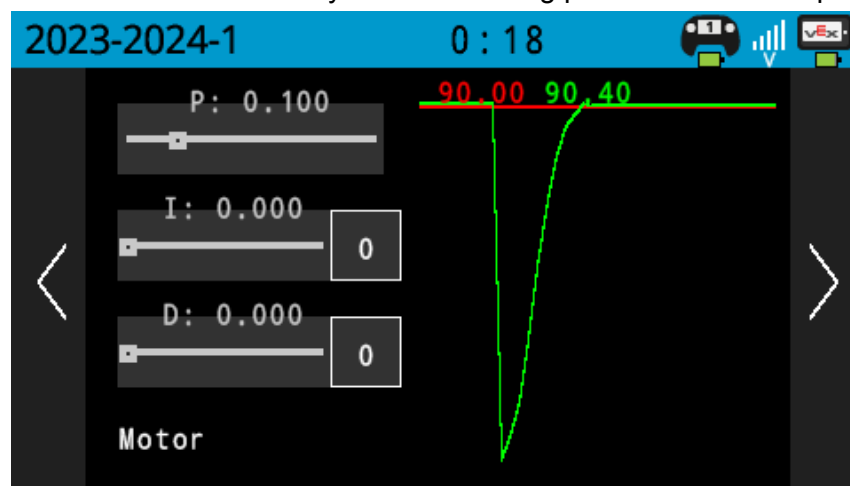
The odometry page has proved incredibly useful in writing and debugging auto and auto skills paths. It shows a picture of the robot on the field as well as a print out of the actual x,y coordinates and heading of the robot. Since we write our autos with respect to the coordinate system of the field,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



A field display for the Over Under season

## PID Tuner

PID controllers are integral to many subsystems on our robot. 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.



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.



Motor Stats screen from our 2023 robot

Cata System Page

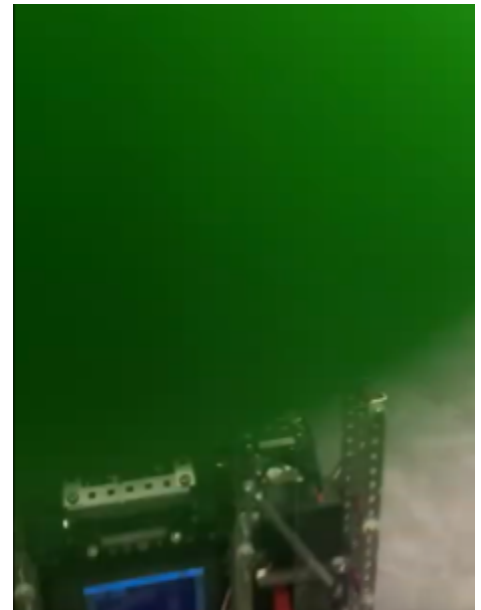
// get a picture of this when we get back

## 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 complexity of the system. This complexity mostly stems from the orchestration of intaking with catapult reloading such that we do not jam our catapult and we 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 catapult 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 what state it is in. Because we have so many sensors, we can determine when intaking would lead to disqualification and simply not honor the intaking message.

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.



Though our system provides thread and disqualification safety, it does not provide programmer safety

Draft work for previous section

- Cata system
- Biggerer and betterer screen system
  - Buttons and sliders enabling hardware to test speeds without a coder redeploying every 5 seconds
  - PID, PIDFF tuners
  - Odometry Map
  - Motor stats
  - Auto chooser
  - Cata State Representation for debugging, tuning
  -
- GPS testing (with all the pretty plots (hell yeah))
- Using GPS to tune odometry
- - ***These pics were from testing it but i think they show a lot about the drift of our wheeled odom***
  - ***Also reasonable to point out that the middle was more accurate than the edges. Especially looking at the 24 row (near the wall but not too far off from drift) to 48 row (farther from the wall but still not too far from drift)***
- GPS odom
  - With our fun filter
  - How it fits with our other odometry components
- Cata system state machine
  - Message passing parallelism 🤖
- Brake mode stuff
  - Velocity brake
  - Smart brake
  - Motivation. Why pure position brake is bad
- Pure pursuit
  - Idrk ask mcgee
- Failed experiments in IMU fusion odom?
  - We cant buy submarine IMUs :(
- Motion controller coolerness + how it interacts with pure pursuit
- Layout system

# Core: Ongoing Projects

~~— N-pod odom~~

- Core-rs - big things cooking
  - Bridge layer if theres anything interesting to say about it
- Vex debug board if we get it working

## Machine Learning

-



# 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



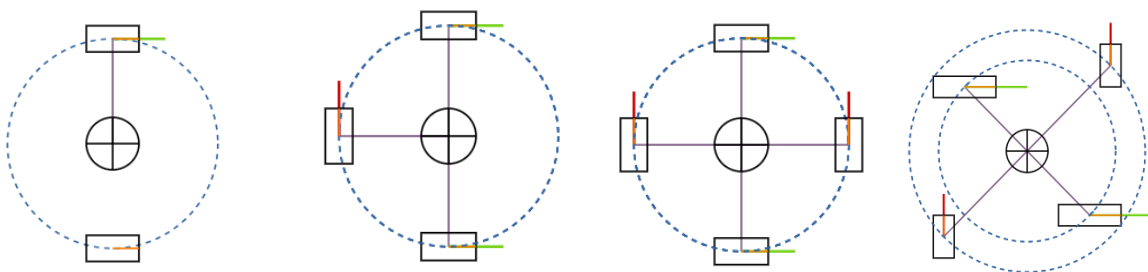
While the transition from C++ to Rust in a robotics context requires investment in terms of learning and codebase modification, the benefits in memory safety, concurrency handling, performance, and maintainability make it a compelling choice. The modern features of Rust, combined with its growing ecosystem 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 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. As well, since tank and 3 wheel odometry are special cases of an n-pod system, we could reduce code duplication.



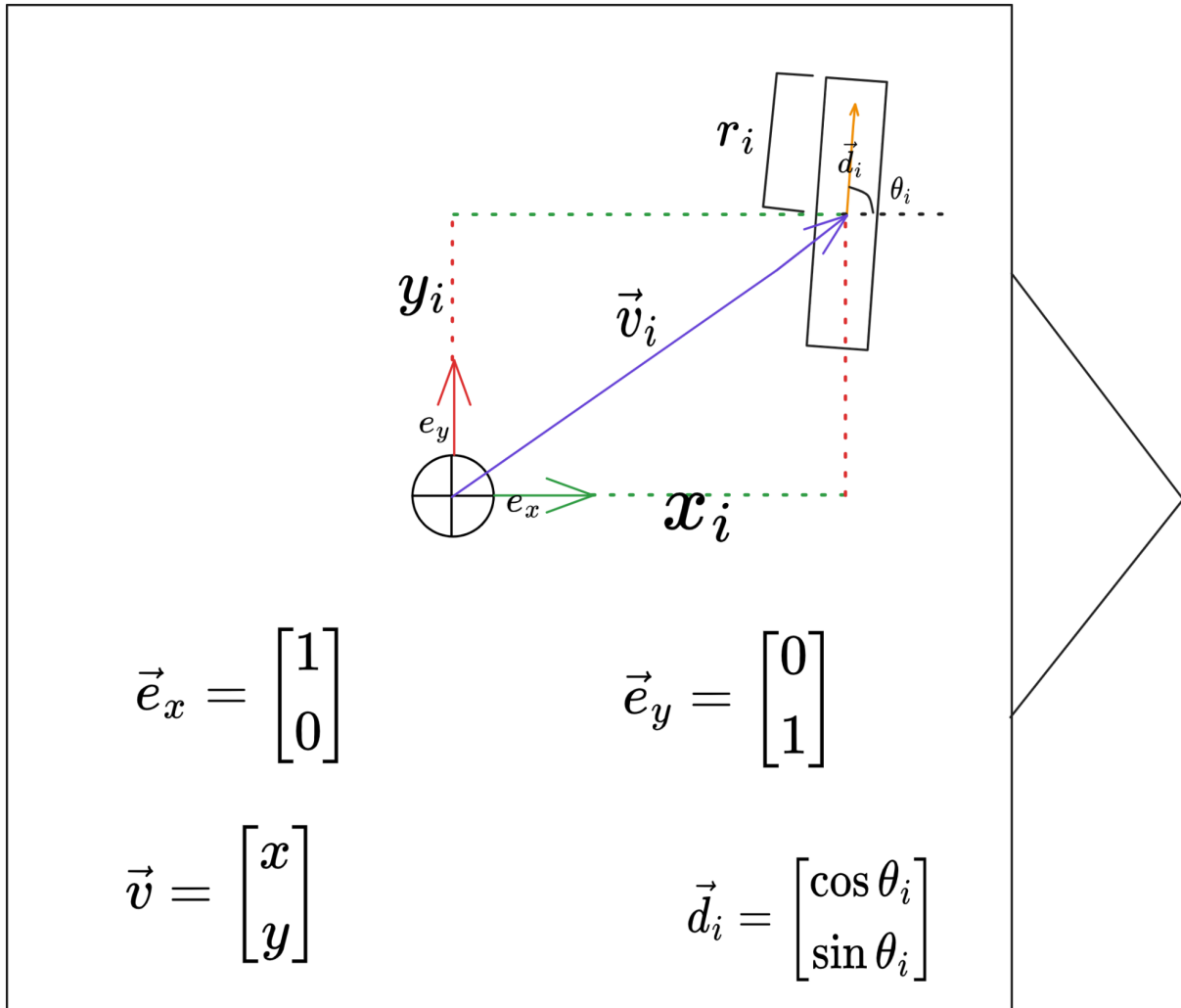
2 pod, 3 pod, and arbitrary pods such a system could handle.

### Syntax

After much brainstorming and many mad scientists 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  $n$  omni tracking wheels. We could read encoder values  $E_1, E_2, \dots, E_n$  from the system in radians from the initial position. As well, each encoder has a

configuration  $(x_1, y_1, \theta_1, r_1), \dots, (x_n, y_n, \theta_n, r_n)$  describing its position relative to the center of rotation, an angle describing its orientation relative to the robot frame, and a radius of the wheel.



The configuration of a tracking wheel on the robot.  $(e_x, e_y)$  are the basis vectors of our coordinate system. X and Y axes of the robot coordinate frame

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 which 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 robot's 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 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  $\vec{e}_x \cdot \vec{d} = \cos(\text{angle between x axis and wheel})$

$F_x$  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,

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

$$F_y$$

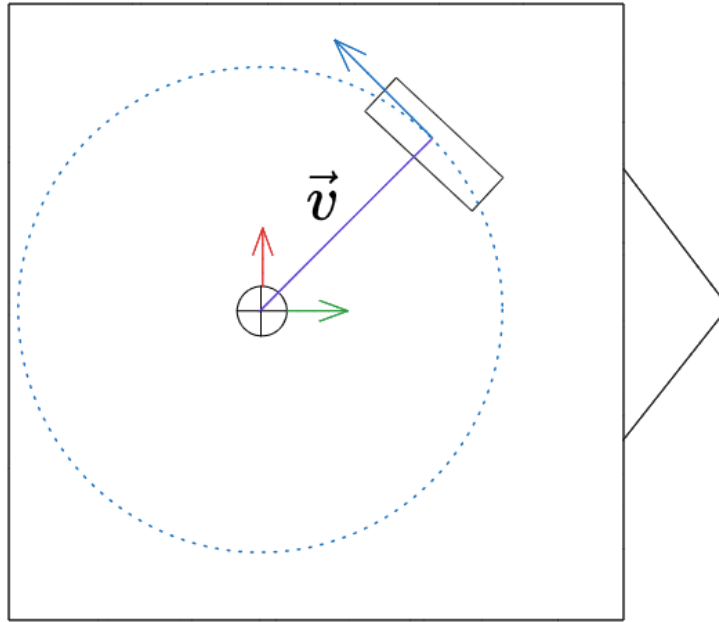
$F_y$  is derived almost identically as  $F_x$  just instead of testing against  $\vec{e}_x$  we test against  $\vec{e}_y$ . So,

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

$$F_\theta$$

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

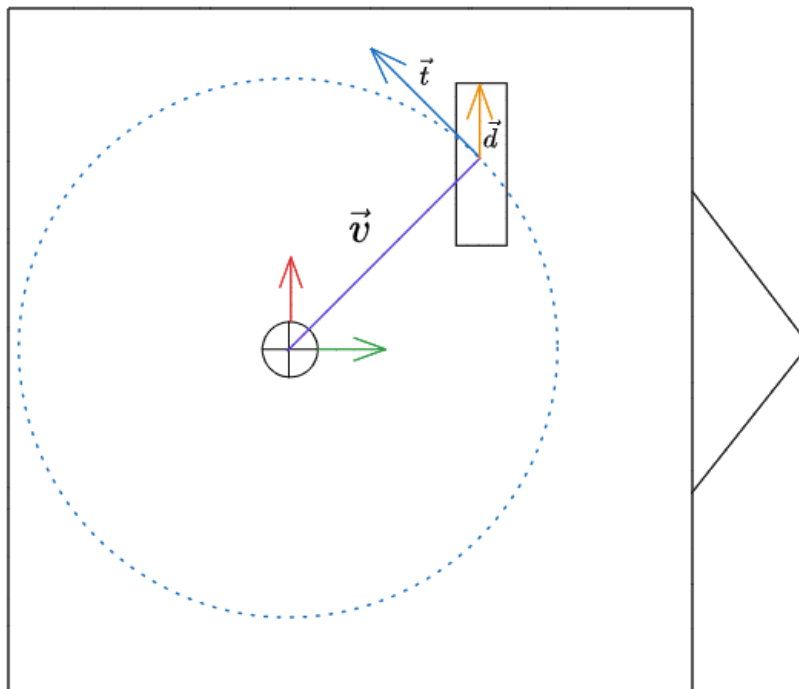


A conceptual perfectly perpendicular wheel

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

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

So

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

## Why factors

These factors make solving this problem much simpler. For the forward case, for some wheel  $i$ , its rotation is the sum of all the motions applied to it. So  $E_i = F_{xi}x + F_{yi}y + F_{\theta i}\theta$ . So, for all the wheels, we plug in the commanded  $(x, y, \theta)$  to each wheel's factors to get its necessary rotation. Since the factors depend only on the wheels 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 solving 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.

$$T = \begin{bmatrix} F_{x1} & F_{y1} & F_{\theta1} \\ \vdots & \vdots & \vdots \\ F_{xn} & F_{yn} & F_{\theta n} \end{bmatrix}$$

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

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

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

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

pose velocity

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

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

encoder wheel  
velocities

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

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

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

The linear algebra behind the solution

# Core: The Funny

Only maybe, this is not professional or something

- 3d renderer
- Video player
- NES (Non-nintendo emulated system) Emulator (did outreach with this cuz a middle schooler was a big fan)

## RIT VEXU Core API

Generated by Doxygen 1.10.0





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

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

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

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

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

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

5.40.3.7 home()	75
5.40.3.8 set_async()	75
5.40.3.9 set_position()	75
5.40.3.10 set_sensor_function()	75
5.40.3.11 set_sensor_reset()	76
5.40.3.12 set_setpoint()	76
5.41 Lift< T >::lift_cfg_t Struct Reference	76
5.41.1 Detailed Description	76
5.42 Logger Class Reference	77
5.42.1 Detailed Description	77
5.42.2 Constructor & Destructor Documentation	77
5.42.2.1 Logger()	77
5.42.3 Member Function Documentation	78
5.42.3.1 Log() [1/2]	78
5.42.3.2 Log() [2/2]	78
5.42.3.3 Logf() [1/2]	78
5.42.3.4 Logf() [2/2]	79
5.42.3.5 LogIn() [1/2]	79
5.42.3.6 LogIn() [2/2]	79
5.43 MotionController::m_profile_cfg_t Struct Reference	79
5.43.1 Detailed Description	80
5.44 Mat2 Struct Reference	80
5.45 MecanumDrive Class Reference	80
5.45.1 Detailed Description	81
5.45.2 Constructor & Destructor Documentation	81
5.45.2.1 MecanumDrive()	81
5.45.3 Member Function Documentation	81
5.45.3.1 auto_drive()	81
5.45.3.2 auto_turn()	82
5.45.3.3 drive()	83
5.45.3.4 drive_raw()	83
5.46 MecanumDrive::mecanumdrive_config_t Struct Reference	84
5.46.1 Detailed Description	84
5.47 motion_t Struct Reference	84
5.47.1 Detailed Description	84
5.48 MotionController Class Reference	85
5.48.1 Detailed Description	85
5.48.2 Constructor & Destructor Documentation	86
5.48.2.1 MotionController()	86
5.48.3 Member Function Documentation	86
5.48.3.1 get()	86
5.48.3.2 get_motion()	86



5.48.3.3	init()	86
5.48.3.4	is_on_target()	87
5.48.3.5	set_limits()	87
5.48.3.6	tune_feedforward()	87
5.48.3.7	update()	88
5.49	MotionControllerPage Class Reference	88
5.49.1	Member Function Documentation	89
5.49.1.1	draw()	89
5.49.1.2	update()	89
5.50	MovingAverage Class Reference	89
5.50.1	Detailed Description	90
5.50.2	Constructor & Destructor Documentation	90
5.50.2.1	MovingAverage() [1/2]	90
5.50.2.2	MovingAverage() [2/2]	90
5.50.3	Member Function Documentation	91
5.50.3.1	add_entry()	91
5.50.3.2	get_size()	91
5.50.3.3	get_value()	91
5.51	Odometry3Wheel Class Reference	92
5.51.1	Detailed Description	93
5.51.2	Constructor & Destructor Documentation	93
5.51.2.1	Odometry3Wheel()	93
5.51.3	Member Function Documentation	94
5.51.3.1	tune()	94
5.51.3.2	update()	94
5.52	Odometry3Wheel::odometry3wheel_cfg_t Struct Reference	95
5.52.1	Detailed Description	95
5.52.2	Member Data Documentation	95
5.52.2.1	off_axis_center_dist	95
5.52.2.2	wheel_diam	95
5.52.2.3	wheelbase_dist	95
5.53	OdometryBase Class Reference	95
5.53.1	Detailed Description	96
5.53.2	Constructor & Destructor Documentation	97
5.53.2.1	OdometryBase()	97
5.53.3	Member Function Documentation	97
5.53.3.1	background_task()	97
5.53.3.2	end_async()	97
5.53.3.3	get_accel()	97
5.53.3.4	get_angular_accel_deg()	98
5.53.3.5	get_angular_speed_deg()	98
5.53.3.6	get_position()	98

5.53.3.7 get_speed()	98
5.53.3.8 pos_diff()	98
5.53.3.9 rot_diff()	99
5.53.3.10 set_position()	99
5.53.3.11 smallest_angle()	99
5.53.3.12 update()	100
5.53.4 Member Data Documentation	100
5.53.4.1 accel	100
5.53.4.2 ang_accel_deg	100
5.53.4.3 ang_speed_deg	100
5.53.4.4 current_pos	101
5.53.4.5 handle	101
5.53.4.6 mut	101
5.53.4.7 speed	101
5.53.4.8 zero_pos	101
5.54 screen::OdometryPage Class Reference	101
5.54.1 Detailed Description	102
5.54.2 Constructor & Destructor Documentation	102
5.54.2.1 OdometryPage()	102
5.54.3 Member Function Documentation	102
5.54.3.1 draw()	102
5.54.3.2 update()	103
5.55 OdometryTank Class Reference	103
5.55.1 Detailed Description	104
5.55.2 Constructor & Destructor Documentation	104
5.55.2.1 OdometryTank() [1/3]	104
5.55.2.2 OdometryTank() [2/3]	105
5.55.2.3 OdometryTank() [3/3]	105
5.55.3 Member Function Documentation	106
5.55.3.1 set_position()	106
5.55.3.2 update()	106
5.56 OdomSetPosition Class Reference	106
5.56.1 Detailed Description	107
5.56.2 Constructor & Destructor Documentation	107
5.56.2.1 OdomSetPosition()	107
5.56.3 Member Function Documentation	108
5.56.3.1 run()	108
5.57 OrCondition Class Reference	108
5.57.1 Member Function Documentation	108
5.57.1.1 test()	108
5.58 screen::Page Class Reference	109
5.58.1 Detailed Description	109

5.58.2 Member Function Documentation	109
5.58.2.1 draw()	109
5.58.2.2 update()	110
5.59 Parallel Class Reference	110
5.59.1 Detailed Description	111
5.59.2 Member Function Documentation	111
5.59.2.1 on_timeout()	111
5.59.2.2 run()	111
5.60 parallel_runner_info Struct Reference	112
5.61 PurePursuit::Path Class Reference	112
5.61.1 Detailed Description	112
5.61.2 Constructor & Destructor Documentation	112
5.61.2.1 Path()	112
5.61.3 Member Function Documentation	113
5.61.3.1 get_points()	113
5.61.3.2 get_radius()	113
5.61.3.3 is_valid()	113
5.62 PID Class Reference	113
5.62.1 Detailed Description	114
5.62.2 Member Enumeration Documentation	114
5.62.2.1 ERROR_TYPE	114
5.62.3 Constructor & Destructor Documentation	114
5.62.3.1 PID()	114
5.62.4 Member Function Documentation	115
5.62.4.1 get()	115
5.62.4.2 get_error()	115
5.62.4.3 get_sensor_val()	115
5.62.4.4 get_target()	116
5.62.4.5 init()	116
5.62.4.6 is_on_target()	116
5.62.4.7 reset()	116
5.62.4.8 set_limits()	117
5.62.4.9 set_target()	117
5.62.4.10 update()	117
5.62.5 Member Data Documentation	118
5.62.5.1 config	118
5.63 PID::pid_config_t Struct Reference	118
5.63.1 Detailed Description	118
5.63.2 Member Data Documentation	118
5.63.2.1 error_method	118
5.63.2.2 on_target_time	119
5.64 PIDFF Class Reference	119

5.64.1 Member Function Documentation	119
5.64.1.1 get()	119
5.64.1.2 init()	120
5.64.1.3 is_on_target()	121
5.64.1.4 set_limits()	121
5.64.1.5 set_target()	121
5.64.1.6 update() [1/2]	122
5.64.1.7 update() [2/2]	122
5.65 screen::PIDPage Class Reference	123
5.65.1 Detailed Description	123
5.65.2 Constructor & Destructor Documentation	123
5.65.2.1 PIDPage()	123
5.65.3 Member Function Documentation	124
5.65.3.1 draw()	124
5.65.3.2 update()	124
5.66 point_t Struct Reference	124
5.66.1 Detailed Description	125
5.66.2 Member Function Documentation	125
5.66.2.1 dist()	125
5.66.2.2 operator+()	125
5.66.2.3 operator-()	125
5.67 pose_t Struct Reference	126
5.67.1 Detailed Description	126
5.68 PurePursuitCommand Class Reference	126
5.68.1 Detailed Description	127
5.68.2 Constructor & Destructor Documentation	127
5.68.2.1 PurePursuitCommand()	127
5.68.3 Member Function Documentation	128
5.68.3.1 on_timeout()	128
5.68.3.2 run()	128
5.69 Rect Struct Reference	128
5.70 RepeatUntil Class Reference	129
5.70.1 Constructor & Destructor Documentation	129
5.70.1.1 RepeatUntil() [1/2]	129
5.70.1.2 RepeatUntil() [2/2]	130
5.70.2 Member Function Documentation	130
5.70.2.1 on_timeout()	130
5.70.2.2 run()	130
5.71 robot_specs_t Struct Reference	131
5.71.1 Detailed Description	131
5.72 screen::ScreenData Struct Reference	131
5.72.1 Detailed Description	132

5.73 screen::ScreenRect Struct Reference . . . . .	132
5.74 Serializer Class Reference . . . . .	132
5.74.1 Detailed Description . . . . .	133
5.74.2 Constructor & Destructor Documentation . . . . .	133
5.74.2.1 Serializer() . . . . .	133
5.74.3 Member Function Documentation . . . . .	133
5.74.3.1 bool_or() . . . . .	133
5.74.3.2 double_or() . . . . .	133
5.74.3.3 int_or() . . . . .	134
5.74.3.4 save_to_disk() . . . . .	134
5.74.3.5 set_bool() . . . . .	134
5.74.3.6 set_double() . . . . .	135
5.74.3.7 set_int() . . . . .	135
5.74.3.8 set_string() . . . . .	135
5.74.3.9 string_or() . . . . .	135
5.75 screen::SizedWidget Struct Reference . . . . .	136
5.76 SliderCfg Struct Reference . . . . .	136
5.77 screen::SliderConfig Struct Reference . . . . .	136
5.78 screen::SliderWidget Class Reference . . . . .	137
5.78.1 Detailed Description . . . . .	137
5.78.2 Constructor & Destructor Documentation . . . . .	137
5.78.2.1 SliderWidget() . . . . .	137
5.78.3 Member Function Documentation . . . . .	137
5.78.3.1 update() . . . . .	137
5.79 SpinRPMCommand Class Reference . . . . .	138
5.79.1 Detailed Description . . . . .	139
5.79.2 Constructor & Destructor Documentation . . . . .	139
5.79.2.1 SpinRPMCommand() . . . . .	139
5.79.3 Member Function Documentation . . . . .	139
5.79.3.1 run() . . . . .	139
5.80 PurePursuit::spline Struct Reference . . . . .	140
5.80.1 Detailed Description . . . . .	140
5.81 screen::StatsPage Class Reference . . . . .	140
5.81.1 Detailed Description . . . . .	141
5.81.2 Constructor & Destructor Documentation . . . . .	141
5.81.2.1 StatsPage() . . . . .	141
5.81.3 Member Function Documentation . . . . .	141
5.81.3.1 draw() . . . . .	141
5.81.3.2 update() . . . . .	141
5.82 TakeBackHalf Class Reference . . . . .	142
5.82.1 Detailed Description . . . . .	142
5.82.2 Member Function Documentation . . . . .	142

5.82.2.1 get()	142
5.82.2.2 init()	142
5.82.2.3 is_on_target()	143
5.82.2.4 set_limits()	143
5.82.2.5 update()	143
5.83 TankDrive Class Reference	144
5.83.1 Detailed Description	145
5.83.2 Member Enumeration Documentation	145
5.83.2.1 BrakeType	145
5.83.3 Constructor & Destructor Documentation	145
5.83.3.1 TankDrive()	145
5.83.4 Member Function Documentation	146
5.83.4.1 drive_arcade()	146
5.83.4.2 drive_forward() [1/2]	146
5.83.4.3 drive_forward() [2/2]	147
5.83.4.4 drive_tank()	148
5.83.4.5 drive_tank_raw()	148
5.83.4.6 drive_to_point() [1/2]	148
5.83.4.7 drive_to_point() [2/2]	149
5.83.4.8 modify_inputs()	150
5.83.4.9 pure_pursuit() [1/2]	150
5.83.4.10 pure_pursuit() [2/2]	151
5.83.4.11 reset_auto()	152
5.83.4.12 stop()	152
5.83.4.13 turn_degrees() [1/2]	152
5.83.4.14 turn_degrees() [2/2]	153
5.83.4.15 turn_to_heading() [1/2]	154
5.83.4.16 turn_to_heading() [2/2]	154
5.84 screen::TextConfig Struct Reference	155
5.85 TimesTestedCondition Class Reference	155
5.85.1 Member Function Documentation	156
5.85.1.1 test()	156
5.86 trapezoid_profile_segment_t Struct Reference	156
5.86.1 Detailed Description	156
5.87 TrapezoidProfile Class Reference	156
5.87.1 Detailed Description	157
5.87.2 Constructor & Destructor Documentation	158
5.87.2.1 TrapezoidProfile()	158
5.87.3 Member Function Documentation	158
5.87.3.1 calculate()	158
5.87.3.2 calculate_time_based()	158
5.87.3.3 get_movement_time()	159

5.87.3.4	set_accel()	159
5.87.3.5	set_endpts()	159
5.87.3.6	set_max_v()	159
5.87.3.7	set_vel_endpts()	160
5.88	TurnDegreesCommand Class Reference	160
5.88.1	Detailed Description	161
5.88.2	Constructor & Destructor Documentation	161
5.88.2.1	TurnDegreesCommand()	161
5.88.3	Member Function Documentation	161
5.88.3.1	on_timeout()	161
5.88.3.2	run()	162
5.89	TurnToHeadingCommand Class Reference	162
5.89.1	Detailed Description	163
5.89.2	Constructor & Destructor Documentation	163
5.89.2.1	TurnToHeadingCommand()	163
5.89.3	Member Function Documentation	163
5.89.3.1	on_timeout()	163
5.89.3.2	run()	163
5.90	Vector2D Class Reference	164
5.90.1	Detailed Description	164
5.90.2	Constructor & Destructor Documentation	164
5.90.2.1	Vector2D() [1/2]	164
5.90.2.2	Vector2D() [2/2]	165
5.90.3	Member Function Documentation	165
5.90.3.1	get_dir()	165
5.90.3.2	get_mag()	165
5.90.3.3	get_x()	165
5.90.3.4	get_y()	166
5.90.3.5	normalize()	166
5.90.3.6	operator*()	166
5.90.3.7	operator+()	166
5.90.3.8	operator-()	167
5.90.3.9	point()	167
5.91	WaitUntilCondition Class Reference	167
5.91.1	Detailed Description	168
5.91.2	Member Function Documentation	168
5.91.2.1	run()	168
5.92	WaitUntilUpToSpeedCommand Class Reference	169
5.92.1	Detailed Description	169
5.92.2	Constructor & Destructor Documentation	169
5.92.2.1	WaitUntilUpToSpeedCommand()	169
5.92.3	Member Function Documentation	170

5.92.3.1 run()	170
5.93 screen::WidgetConfig Struct Reference	170
5.94 screen::WidgetPage Class Reference	171
5.94.1 Member Function Documentation	171
5.94.1.1 draw()	171
5.94.1.2 update()	171
<b>6 File Documentation</b>	<b>173</b>
6.1 robot_specs.h	173
6.2 custom_encoder.h	173
6.3 flywheel.h	174
6.4 layout.h	174
6.5 lift.h	175
6.6 mecanum_drive.h	177
6.7 odometry_3wheel.h	178
6.8 odometry_base.h	179
6.9 odometry_tank.h	179
6.10 screen.h	180
6.11 tank_drive.h	183
6.12 auto_chooser.h	184
6.13 auto_command.h	185
6.14 basic_command.h	187
6.15 command_controller.h	187
6.16 delay_command.h	188
6.17 drive_commands.h	188
6.18 flywheel_commands.h	190
6.19 bang_bang.h	191
6.20 feedback_base.h	191
6.21 feedforward.h	192
6.22 motion_controller.h	192
6.23 pid.h	193
6.24 pidff.h	194
6.25 take_back_half.h	194
6.26 trapezoid_profile.h	195
6.27 generic_auto.h	196
6.28 geometry.h	196
6.29 graph_drawer.h	197
6.30 logger.h	198
6.31 math_util.h	198
6.32 moving_average.h	199
6.33 pure_pursuit.h	200
6.34 serializer.h	201



6.35 <a href="#">vector2d.h</a> . . . . .	202
<b>Index</b>	<b>203</b>

# Chapter 1

## Core

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

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

### 1.1 Getting Started

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

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

### 1.2 Features

Here is the current feature list this repo provides:

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

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

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

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



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

AutoCommand . . . . .	18
Async . . . . .	14
BasicSolenoidSet . . . . .	22
BasicSpinCommand . . . . .	23
BasicStopCommand . . . . .	25
Branch . . . . .	27
DelayCommand . . . . .	36
DriveForwardCommand . . . . .	38
DriveStopCommand . . . . .	40
DriveToPointCommand . . . . .	42
FlywheelStopCommand . . . . .	58
FlywheelStopMotorsCommand . . . . .	59
FlywheelStopNonTasksCommand . . . . .	61
FunctionCommand . . . . .	62
InOrder . . . . .	70
OdomSetPosition . . . . .	106
Parallel . . . . .	110
PurePursuitCommand . . . . .	126
RepeatUntil . . . . .	129
SpinRPMCommand . . . . .	138
TurnDegreesCommand . . . . .	160
TurnToHeadingCommand . . . . .	162
WaitUntilCondition . . . . .	167
WaitUntilUpToSpeedCommand . . . . .	169
screen::ButtonConfig . . . . .	28
screen::ButtonWidget . . . . .	28
screen::CheckboxConfig . . . . .	30
CommandController . . . . .	30
Condition . . . . .	33
AndCondition . . . . .	13
FunctionCondition . . . . .	63
IfTimePassed . . . . .	69
OrCondition . . . . .	108
TimesTestedCondition . . . . .	155
vex::encoder	

CustomEncoder	34
AutoChooser::entry_t	44
Feedback	47
BangBang	20
MotionController	85
PID	113
PIDFF	119
TakeBackHalf	142
FeedForward	49
FeedForward::ff_config_t	51
Filter	52
ExponentialMovingAverage	45
MovingAverage	89
Flywheel	53
GenericAuto	65
GraphDrawer	67
PurePursuit::hermite_point	69
screen::LabelConfig	72
Lift< T >	72
Lift< T >::lift_cfg_t	76
Logger	77
MotionController::m_profile_cfg_t	79
Mat2	80
MecanumDrive	80
MecanumDrive::mecanumdrive_config_t	84
motion_t	84
Odometry3Wheel::odometry3wheel_cfg_t	95
OdometryBase	95
Odometry3Wheel	92
OdometryTank	103
screen::Page	109
AutoChooser	15
FlywheelPage	57
MotionControllerPage	88
screen::FunctionPage	64
screen::OdometryPage	101
screen::PIDPage	123
screen::StatsPage	140
screen::WidgetPage	171
parallel_runner_info	112
PurePursuit::Path	112
PID::pid_config_t	118
point_t	124
pose_t	126
Rect	128
robot_specs_t	131
screen::ScreenData	131
screen::ScreenRect	132
Serializer	132
screen::SizedWidget	136
SliderCfg	136
screen::SliderConfig	136
screen::SliderWidget	137
PurePursuit::spline	140
TankDrive	144
screen::TextConfig	155
trapezoid_profile_segment_t	156

TrapezoidProfile . . . . .	156
Vector2D . . . . .	164
screen::WidgetConfig . . . . .	170



## Chapter 3

# Class Index

### 3.1 Class List

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

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



<a href="#">FlywheelStopNonTasksCommand</a>	61
<a href="#">FunctionCommand</a>	62
<a href="#">FunctionCondition</a>	
<a href="#">FunctionCondition</a> is a quick and dirty <a href="#">Condition</a> to wrap some expression that should be evaluated at runtime	63
<a href="#">screen::FunctionPage</a>	
Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything	64
<a href="#">GenericAuto</a>	65
<a href="#">GraphDrawer</a>	67
<a href="#">PurePursuit::hermite_point</a>	69
<a href="#">IfTimePassed</a>	
<a href="#">IfTimePassed</a> tests based on time since the command controller was constructed. Returns true if elapsed time > time_s	69
<a href="#">InOrder</a>	
<a href="#">InOrder</a> runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timeouts?	70
<a href="#">screen::LabelConfig</a>	72
<a href="#">Lift&lt; T &gt;</a>	72
<a href="#">Lift&lt; T &gt;::lift_cfg_t</a>	76
<a href="#">Logger</a>	
Class to simplify writing to files	77
<a href="#">MotionController::m_profile_cfg_t</a>	79
<a href="#">Mat2</a>	80
<a href="#">MecanumDrive</a>	80
<a href="#">MecanumDrive::mecanumdrive_config_t</a>	84
<a href="#">motion_t</a>	84
<a href="#">MotionController</a>	85
<a href="#">MotionControllerPage</a>	88
<a href="#">MovingAverage</a>	89
<a href="#">Odometry3Wheel</a>	92
<a href="#">Odometry3Wheel::odometry3wheel_cfg_t</a>	95
<a href="#">OdometryBase</a>	95
<a href="#">screen::OdometryPage</a>	
Page that shows odometry position and rotation and a map (if an sd card with the file is on)	101
<a href="#">OdometryTank</a>	103
<a href="#">OdomSetPosition</a>	106
<a href="#">OrCondition</a>	108
<a href="#">screen::Page</a>	
Page describes one part of the screen slideshow	109
<a href="#">Parallel</a>	
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	110
<a href="#">parallel_runner_info</a>	112
<a href="#">PurePursuit::Path</a>	112
<a href="#">PID</a>	113
<a href="#">PID::pid_config_t</a>	118
<a href="#">PIDFF</a>	119
<a href="#">screen::PIDPage</a>	
PIDPage provides a way to tune a pid controller on the screen	123
<a href="#">point_t</a>	124
<a href="#">pose_t</a>	126
<a href="#">PurePursuitCommand</a>	126
<a href="#">Rect</a>	128
<a href="#">RepeatUntil</a>	129
<a href="#">robot_specs_t</a>	131
<a href="#">screen::ScreenData</a>	
Holds the data that will be passed to the screen thread you probably shouldnt have to use it	131

screen::ScreenRect	132
Serializer	
Serializes Arbitrary data to a file on the SD Card	132
screen::SizedWidget	136
SliderCfg	136
screen::SliderConfig	136
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	137
SpinRPMCommand	138
PurePursuit::spline	140
screen::StatsPage	
Draws motor stats and battery stats to the screen	140
TakeBackHalf	
A velocity controller	142
TankDrive	144
screen::TextConfig	155
TimesTestedCondition	155
trapezoid_profile_segment_t	156
TrapezoidProfile	156
TurnDegreesCommand	160
TurnToHeadingCommand	162
Vector2D	164
WaitUntilCondition	
Waits until the condition is true	167
WaitUntilUpToSpeedCommand	169
screen::WidgetConfig	170
screen::WidgetPage	171



# Chapter 4

## File Index

### 4.1 File List

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

include/robot_specs.h	173
include/subsystems/custom_encoder.h	173
include/subsystems/flywheel.h	174
include/subsystems/layout.h	174
include/subsystems/lift.h	175
include/subsystems/mecanum_drive.h	177
include/subsystems/screen.h	180
include/subsystems/tank_drive.h	183
include/subsystems/odometry/odometry_3wheel.h	178
include/subsystems/odometry/odometry_base.h	179
include/subsystems/odometry/odometry_tank.h	179
include/utils/auto_chooser.h	184
include/utils/generic_auto.h	196
include/utils/geometry.h	196
include/utils/graph_drawer.h	197
include/utils/logger.h	198
include/utils/math_util.h	198
include/utils/moving_average.h	199
include/utils/pure_pursuit.h	200
include/utils/serializer.h	201
include/utils/vector2d.h	202
include/utils/command_structure/auto_command.h	185
include/utils/command_structure/basic_command.h	187
include/utils/command_structure/command_controller.h	187
include/utils/command_structure/delay_command.h	188
include/utils/command_structure/drive_commands.h	188
include/utils/command_structure/flywheel_commands.h	190
include/utils/controls/bang_bang.h	191
include/utils/controls/feedback_base.h	191
include/utils/controls/feedforward.h	192
include/utils/controls/motion_controller.h	192
include/utils/controls/pid.h	193
include/utils/controls/pidff.h	194
include/utils/controls/take_back_half.h	194
include/utils/controls/trapezoid_profile.h	195

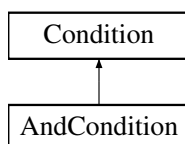


## Chapter 5

# Class Documentation

### 5.1 AndCondition Class Reference

Inheritance diagram for AndCondition:



#### Public Member Functions

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

#### Public Member Functions inherited from [Condition](#)

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

#### 5.1.1 Member Function Documentation

##### 5.1.1.1 [test\(\)](#)

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

Implements [Condition](#).

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

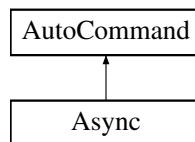
- src/utls/command\_structure/auto\_command.cpp

## 5.2 Async Class Reference

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

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

Inheritance diagram for Async:



### Public Member Functions

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

### Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* **withTimeout** (double t\_seconds)
- [AutoCommand](#) \* **withCancelCondition** ([Condition](#) \*true\_to\_end)

### Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default\_timeout** = 10.0

#### 5.2.1 Detailed Description

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

## 5.2.2 Member Function Documentation

### 5.2.2.1 run()

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

Executes the command Overridden by child classes

#### Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

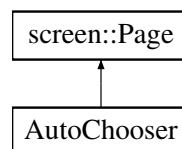
The documentation for this class was generated from the following files:

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

## 5.3 AutoChooser Class Reference

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

Inheritance diagram for AutoChooser:



### Classes

- struct [entry\\_t](#)

### Public Member Functions

- [AutoChooser](#) (std::vector< std::string > paths, size\_t def=0)
- void [update](#) (bool was\_pressed, int x, int y)  
*collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this Page (only drawn page gets touch updates))*
- void [draw](#) (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number)  
*draw stored data to the screen (runs at 10 hz and only runs if this page is in front)*
- size\_t [get\\_choice](#) ()

### Protected Attributes

- size\_t [choice](#)
- std::vector< [entry\\_t](#) > [list](#)



## Static Protected Attributes

- static const size\_t **width** = 380
- static const size\_t **height** = 220

### 5.3.1 Detailed Description

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

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 AutoChooser()

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

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

#### Parameters

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

### 5.3.3 Member Function Documentation

#### 5.3.3.1 draw()

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

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

#### Parameters

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

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

### 5.3.3.2 get\_choice()

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

Get the currently selected auto choice

#### Returns

the identifier to the auto path

Return the selected autonomous

### 5.3.3.3 update()

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

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

#### Parameters

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

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

## 5.3.4 Member Data Documentation

### 5.3.4.1 choice

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

the current choice of auto

### 5.3.4.2 list

```
std::vector<entry_t> AutoChooser::list [protected]
```

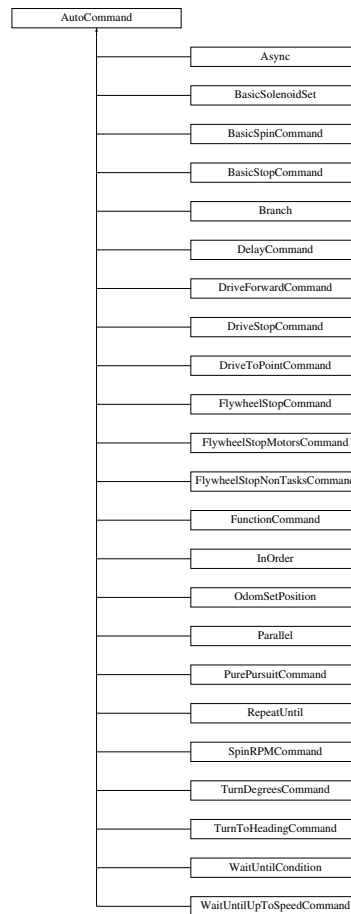
< a list of all possible auto choices

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

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

## 5.4 AutoCommand Class Reference

Inheritance diagram for AutoCommand:



### Public Member Functions

- virtual bool [run](#) ()
- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* **withTimeout** (double t\_seconds)
- [AutoCommand](#) \* **withCancelCondition** ([Condition](#) \*true\_to\_end)

### Public Attributes

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

### Static Public Attributes

- static constexpr double **default\_timeout** = 10.0

## 5.4.1 Member Function Documentation

### 5.4.1.1 on\_timeout()

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

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

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

### 5.4.1.2 run()

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

Executes the command Overridden by child classes

#### Returns

true when the command is finished, false otherwise

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

## 5.4.2 Member Data Documentation

### 5.4.2.1 timeout\_seconds

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

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

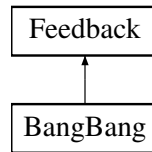
- a drive command hitting a wall and not being able to reach its target
- a command that waits until something is up to speed that never gets up to speed because of battery voltage
- something else...

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

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

## 5.5 BangBang Class Reference

Inheritance diagram for BangBang:



### Public Member Functions

- **BangBang** (double threshold, double low, double high)
- void **init** (double start\_pt, double set\_pt, double start\_vel=0.0, double end\_vel=0.0) override
- double **update** (double val) override
- double **get** () override
- void **set\_limits** (double lower, double upper) override
- bool **is\_on\_target** () override

### 5.5.1 Member Function Documentation

#### 5.5.1.1 get()

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

#### Returns

the last saved result from the feedback controller

Implements [Feedback](#).

#### 5.5.1.2 init()

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

Initialize the feedback controller for a movement

#### Parameters

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

Implements [Feedback](#).

### 5.5.1.3 is\_on\_target()

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

#### Returns

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

### 5.5.1.4 set\_limits()

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

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

#### Parameters

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

Implements [Feedback](#).

### 5.5.1.5 update()

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

Iterate the feedback loop once with an updated sensor value

#### Parameters

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

#### Returns

feedback loop result

Implements [Feedback](#).

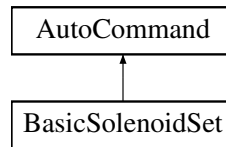
The documentation for this class was generated from the following files:

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

## 5.6 BasicSolenoidSet Class Reference

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

Inheritance diagram for BasicSolenoidSet:



### Public Member Functions

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

### Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

### Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

#### 5.6.1 Detailed Description

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

#### 5.6.2 Constructor & Destructor Documentation

##### 5.6.2.1 BasicSolenoidSet()

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

Construct a new [BasicSolenoidSet](#) Command.

## Parameters

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

### 5.6.3 Member Function Documentation

#### 5.6.3.1 run()

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

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

## Returns

True Command runs once

Reimplemented from [AutoCommand](#).

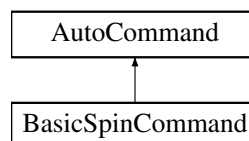
The documentation for this class was generated from the following files:

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

## 5.7 BasicSpinCommand Class Reference

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

Inheritance diagram for BasicSpinCommand:



### Public Types

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

### Public Member Functions

- [BasicSpinCommand](#) (vex::motor &motor, vex::directionType dir, BasicSpinCommand::type setting, double power)  
Construct a new [BasicSpinCommand](#).
- bool [run](#) () override  
Runs the [BasicSpinCommand](#) Overrides run from Auto Command.



## Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

## Additional Inherited Members

## Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.7.1 Detailed Description

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

### 5.7.2 Constructor & Destructor Documentation

#### 5.7.2.1 BasicSpinCommand()

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

Construct a new [BasicSpinCommand](#).

a BasicMotorSpin Command

#### Parameters

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

### 5.7.3 Member Function Documentation

#### 5.7.3.1 run()

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

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

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

#### Returns

- True [Async](#) running command
- True Command runs once

Reimplemented from [AutoCommand](#).

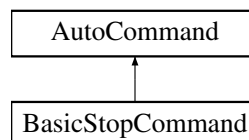
The documentation for this class was generated from the following files:

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

## 5.8 BasicStopCommand Class Reference

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

Inheritance diagram for BasicStopCommand:



#### Public Member Functions

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

#### Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

## Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

## 5.8.1 Detailed Description

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

## 5.8.2 Constructor & Destructor Documentation

### 5.8.2.1 BasicStopCommand()

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

Construct a new BasicMotorStop Command.

Construct a BasicMotorStop Command.

#### Parameters

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

## 5.8.3 Member Function Documentation

### 5.8.3.1 run()

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

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

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

**Returns**

True Command runs once

Reimplemented from [AutoCommand](#).

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

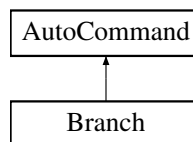
- include/utils/command\_structure/basic\_command.h
- src/utils/command\_structure/basic\_command.cpp

## 5.9 Branch Class Reference

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

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

Inheritance diagram for Branch:

**Public Member Functions**

- **Branch** ([Condition](#) \*cond, [AutoCommand](#) \*false\_choice, [AutoCommand](#) \*true\_choice)
- bool [run](#) () override
- void [on\\_timeout](#) () override

**Public Member Functions inherited from [AutoCommand](#)**

- [AutoCommand](#) \* **withTimeout** (double t\_seconds)
- [AutoCommand](#) \* **withCancelCondition** ([Condition](#) \*true\_to\_end)

**Additional Inherited Members****Public Attributes inherited from [AutoCommand](#)**

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* **true\_to\_end** = nullptr

**Static Public Attributes inherited from [AutoCommand](#)**

- static constexpr double **default\_timeout** = 10.0

### 5.9.1 Detailed Description

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

### 5.9.2 Member Function Documentation

#### 5.9.2.1 `on_timeout()`

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

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

Reimplemented from [AutoCommand](#).

#### 5.9.2.2 `run()`

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

Executes the command. Overridden by child classes

Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

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

- `include/Utils/Command_Structure/auto_command.h`
- `src/Utils/Command_Structure/auto_command.cpp`

## 5.10 `screen::ButtonConfig` Struct Reference

**Public Attributes**

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

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

- `include/subsystems/screen.h`

## 5.11 `screen::ButtonWidget` Class Reference

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

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

## Public Member Functions

- [ButtonWidget](#) (std::function< void(void)> onpress, [Rect](#) rect, std::string name)  
*Create a Button widget.*
- [ButtonWidget](#) (void(\*onpress)(), [Rect](#) rect, std::string name)  
*Create a Button widget.*
- bool [update](#) (bool was\_pressed, int x, int y)  
*responds to user input*
- void [draw](#) (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number)  
*draws the button to the screen*

### 5.11.1 Detailed Description

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

### 5.11.2 Constructor & Destructor Documentation

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

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

Create a Button widget.

##### Parameters

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

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

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

Create a Button widget.

##### Parameters

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

### 5.11.3 Member Function Documentation

#### 5.11.3.1 update()

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

responds to user input

##### Parameters

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

##### Returns

true if the button was pressed

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

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

## 5.12 screen::CheckboxConfig Struct Reference

### Public Attributes

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

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

- include/subsystems/screen.h

## 5.13 CommandController Class Reference

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

## Public Member Functions

- **CommandController ()**  
Create an empty [CommandController](#). Add Command with [CommandController::add\(\)](#)
- **CommandController (std::initializer\_list< [AutoCommand](#) \* > cmds)**  
Create a [CommandController](#) with commands pre added. More can be added with [CommandController::add\(\)](#)
- void **add** (std::vector< [AutoCommand](#) \* > cmds)
- void **add** ([AutoCommand](#) \*cmd, double timeout\_seconds=10.0)
- void **add** (std::vector< [AutoCommand](#) \* > cmds, double timeout\_sec)
- void **add\_delay** (int ms)
- void **add\_cancel\_func** (std::function< bool(void)> true\_if\_cancel)  
*add\_cancel\_func specifies that when this func evaluates to true, to cancel the command controller*
- void **run** ()
- bool **last\_command\_timed\_out** ()

### 5.13.1 Detailed Description

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

### 5.13.2 Constructor & Destructor Documentation

#### 5.13.2.1 CommandController()

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

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

#### Parameters

<a href="#">cmds</a>	
----------------------	--

### 5.13.3 Member Function Documentation

#### 5.13.3.1 add() [1/3]

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

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

#### Parameters

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



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

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

Adds a command to the queue

**Parameters**

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

Add multiple commands to the queue. No timeout here.

**Parameters**

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

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

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

Add multiple commands to the queue. No timeout here.

**Parameters**

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

Add multiple commands to the queue. No timeout here.

**Parameters**

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

**5.13.3.4 add\_cancel\_func()**

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

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

## Parameters

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

**5.13.3.5 add\_delay()**

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

Adds a command that will delay progression of the queue

## Parameters

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

**5.13.3.6 last\_command\_timed\_out()**

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

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

## Returns

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

**5.13.3.7 run()**

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

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

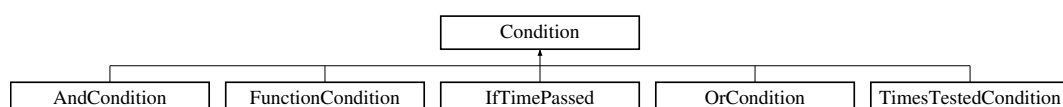
The documentation for this class was generated from the following files:

- `include/utls/command_structure/command_controller.h`
- `src/utls/command_structure/command_controller.cpp`

**5.14 Condition Class Reference**

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

Inheritance diagram for Condition:



## Public Member Functions

- [Condition](#) \* **Or** ([Condition](#) \*b)
- [Condition](#) \* **And** ([Condition](#) \*b)
- virtual bool **test** ()=0

### 5.14.1 Detailed Description

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

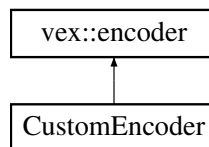
The documentation for this class was generated from the following files:

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

## 5.15 CustomEncoder Class Reference

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

Inheritance diagram for CustomEncoder:



## Public Member Functions

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

### 5.15.1 Detailed Description

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

### 5.15.2 Constructor & Destructor Documentation

#### 5.15.2.1 CustomEncoder()

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

Construct an encoder with a custom number of ticks

## Parameters

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

### 5.15.3 Member Function Documentation

#### 5.15.3.1 position()

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

get the position that the encoder is at

## Parameters

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

## Returns

the position of the encoder in the units specified

#### 5.15.3.2 rotation()

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

get the rotation that the encoder is at

## Parameters

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

## Returns

the rotation of the encoder in the units specified

#### 5.15.3.3 setPosition()

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

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

**Parameters**

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

**5.15.3.4 setRotation()**

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

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

**Parameters**

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

**5.15.3.5 velocity()**

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

get the velocity that the encoder is moving at

**Parameters**

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

**Returns**

the velocity of the encoder in the units specified

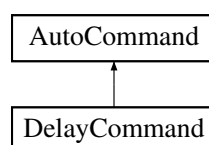
The documentation for this class was generated from the following files:

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

**5.16 DelayCommand Class Reference**

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

Inheritance diagram for DelayCommand:



## Public Member Functions

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

## Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

## Additional Inherited Members

## Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.16.1 Detailed Description

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

### 5.16.2 Constructor & Destructor Documentation

#### 5.16.2.1 DelayCommand()

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

Construct a delay command

#### Parameters

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

### 5.16.3 Member Function Documentation

#### 5.16.3.1 run()

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

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

#### Returns

true when complete

Reimplemented from [AutoCommand](#).

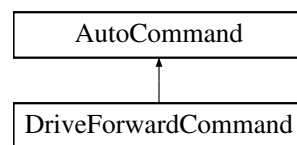
The documentation for this class was generated from the following file:

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

## 5.17 DriveForwardCommand Class Reference

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

Inheritance diagram for DriveForwardCommand:



#### Public Member Functions

- [DriveForwardCommand](#) ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, double inches, directionType dir, double max\_speed=1, double end\_speed=0)
- bool [run](#) () override
- void [on\\_timeout](#) () override

#### Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

#### Additional Inherited Members

#### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

#### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.17.1 Detailed Description

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

### 5.17.2 Constructor & Destructor Documentation

#### 5.17.2.1 DriveForwardCommand()

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

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

Currently includes:

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

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

Currently includes:

- `set_position` Construct a DriveForward Command

#### Parameters

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

### 5.17.3 Member Function Documentation

#### 5.17.3.1 on\_timeout()

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



Cleans up drive system if we time out before finishing

reset the drive system if we timeout

Reimplemented from [AutoCommand](#).

### 5.17.3.2 run()

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

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

#### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

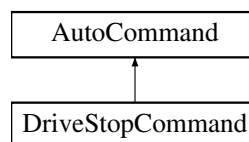
The documentation for this class was generated from the following files:

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

## 5.18 DriveStopCommand Class Reference

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

Inheritance diagram for DriveStopCommand:



#### Public Member Functions

- [DriveStopCommand](#) ([TankDrive](#) &drive\_sys)
- bool [run](#) () override
- void [on\\_timeout](#) () override

#### Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

## Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

## 5.18.1 Detailed Description

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

## 5.18.2 Constructor & Destructor Documentation

### 5.18.2.1 DriveStopCommand()

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

Construct a DriveStop Command

#### Parameters

<a href="#">drive_sys</a>	the drive system we are commanding
---------------------------	------------------------------------

## 5.18.3 Member Function Documentation

### 5.18.3.1 on\_timeout()

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

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

Reimplemented from [AutoCommand](#).

### 5.18.3.2 run()

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

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

**Returns**

true when execution is complete, false otherwise

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

**Returns**

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

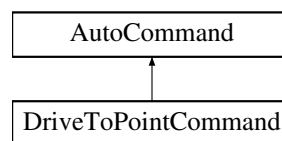
The documentation for this class was generated from the following files:

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

## 5.19 DriveToPointCommand Class Reference

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

Inheritance diagram for DriveToPointCommand:

**Public Member Functions**

- [DriveToPointCommand](#) ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, double x, double y, directionType dir, double max\_speed=1, double end\_speed=0)
- [DriveToPointCommand](#) ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, [point\\_t](#) point, directionType dir, double max\_speed=1, double end\_speed=0)
- bool [run](#) () override

**Public Member Functions inherited from [AutoCommand](#)**

- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

**Additional Inherited Members****Public Attributes inherited from [AutoCommand](#)**

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default\_timeout** = 10.0

### 5.19.1 Detailed Description

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

### 5.19.2 Constructor & Destructor Documentation

#### 5.19.2.1 DriveToPointCommand() [1/2]

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

Construct a DriveForward Command

##### Parameters

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

#### 5.19.2.2 DriveToPointCommand() [2/2]

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

Construct a DriveForward Command

##### Parameters

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

### 5.19.3 Member Function Documentation

#### 5.19.3.1 run()

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

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

##### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

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

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

## 5.20 AutoChooser::entry\_t Struct Reference

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

### Public Attributes

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

#### 5.20.1 Detailed Description

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

### 5.20.2 Member Data Documentation

#### 5.20.2.1 name

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

name of the auto represented by the block

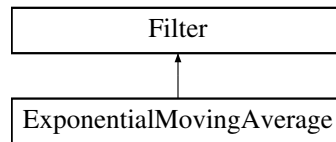
The documentation for this struct was generated from the following file:

- include/utls/auto\_chooser.h

## 5.21 ExponentialMovingAverage Class Reference

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

Inheritance diagram for ExponentialMovingAverage:



### Public Member Functions

- [ExponentialMovingAverage](#) (int buffer\_size)
- [ExponentialMovingAverage](#) (int buffer\_size, double starting\_value)
- void [add\\_entry](#) (double n) override
- double [get\\_value](#) () const override
- int [get\\_size](#) ()

### 5.21.1 Detailed Description

#### [ExponentialMovingAverage](#)

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

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

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

### 5.21.2 Constructor & Destructor Documentation

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

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

Create a moving average calculator with 0 as the default value

#### Parameters

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

### 5.21.2.2 ExponentialMovingAverage() [2/2]

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

Create a moving average calculator with a specified default value

#### Parameters

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

## 5.21.3 Member Function Documentation

### 5.21.3.1 add\_entry()

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

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

#### Parameters

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

Implements [Filter](#).

### 5.21.3.2 get\_size()

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

How many samples the average is made from

#### Returns

the number of samples used to calculate this average

### 5.21.3.3 get\_value()

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

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

**Returns**

the calculated average. `sum(samples)/numsamples`

How many samples the average is made from

**Returns**

the number of samples used to calculate this average

Implements [Filter](#).

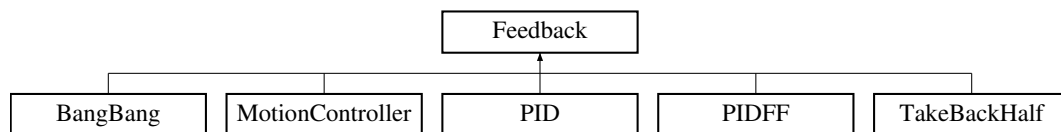
The documentation for this class was generated from the following files:

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

## 5.22 Feedback Class Reference

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

Inheritance diagram for Feedback:

**Public Member Functions**

- virtual void [init](#) (double start\_pt, double set\_pt, double start\_vel=0.0, double end\_vel=0.0)=0
- virtual double [update](#) (double val)=0
- virtual double [get](#) ()=0
- virtual void [set\\_limits](#) (double lower, double upper)=0
- virtual bool [is\\_on\\_target](#) ()=0

### 5.22.1 Detailed Description

Interface so that subsystems can easily switch between feedback loops

**Author**

Ryan McGee

**Date**

9/25/2022



## 5.22.2 Member Function Documentation

### 5.22.2.1 `get()`

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

#### Returns

the last saved result from the feedback controller

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

### 5.22.2.2 `init()`

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

Initialize the feedback controller for a movement

#### Parameters

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

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

### 5.22.2.3 `is_on_target()`

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

#### Returns

true if the feedback controller has reached it's setpoint

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

### 5.22.2.4 `set_limits()`

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

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

## Parameters

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

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

**5.22.2.5 update()**

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

Iterate the feedback loop once with an updated sensor value

## Parameters

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

## Returns

feedback loop result

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

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

- `include/utils/controls/feedback_base.h`

**5.23 FeedForward Class Reference**

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

**Classes**

- struct [ff\\_config\\_t](#)

**Public Member Functions**

- [FeedForward](#) ([ff\\_config\\_t](#) &cfg)
- double [calculate](#) (double v, double a, double pid\_ref=0.0)  
*Perform the feedforward calculation.*

### 5.23.1 Detailed Description

#### FeedForward

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

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

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

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

#### Author

Ryan McGee

#### Date

6/13/2022

### 5.23.2 Constructor & Destructor Documentation

#### 5.23.2.1 FeedForward()

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

Creates a [FeedForward](#) object.

#### Parameters

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

### 5.23.3 Member Function Documentation

#### 5.23.3.1 calculate()

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

Perform the feedforward calculation.

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

## Parameters

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

## Returns

A feedforward that should closely represent the system if tuned correctly

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

- include/Utils/controls/feedforward.h

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

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

## Public Attributes

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

### 5.24.1 Detailed Description

*ff\_config\_t* holds the parameters to make the theoretical model of a real world system equation is of the form  $kS$  if the system is not stopped, 0 otherwise

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

### 5.24.2 Member Data Documentation

#### 5.24.2.1 *kA*

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

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

### 5.24.2.2 kG

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

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

### 5.24.2.3 kS

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

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

### 5.24.2.4 kV

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

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

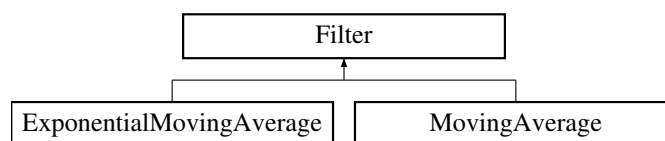
The documentation for this struct was generated from the following file:

- include/utlis/controls/feedforward.h

## 5.25 Filter Class Reference

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

Inheritance diagram for Filter:



### Public Member Functions

- virtual void [add\\_entry](#) (double n)=0
- virtual double [get\\_value](#) () const =0

### 5.25.1 Detailed Description

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

## 5.25.2 Member Function Documentation

### 5.25.2.1 add\_entry()

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

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

### 5.25.2.2 get\_value()

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

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

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

- include/utls/moving\_average.h

## 5.26 Flywheel Class Reference

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

### Public Member Functions

- [Flywheel](#) (vex::motor\_group &motors, [Feedback](#) &feedback, [FeedForward](#) &helper, const double ratio, [Filter](#) &filt)
- double [get\\_target](#) () const
- double [getRPM](#) () const
- vex::motor\_group & [get\\_motors](#) () const
- void [spin\\_manual](#) (double speed, directionType dir=fwd)
- void [spin\\_rpm](#) (double rpm)
- void [stop](#) ()
- bool [is\\_on\\_target](#) ()  
*check if the feedback controller thinks the flywheel is on target*
- [screen::Page](#) \* [Page](#) () const  
*Creates a page displaying info about the flywheel.*
- [AutoCommand](#) \* [SpinRpmCmd](#) (int rpm)  
*Creates a new auto command to spin the flywheel at the desired velocity.*
- [AutoCommand](#) \* [WaitUntilUpToSpeedCmd](#) ()  
*Creates a new auto command that will hold until the flywheel has its target as defined by its feedback controller.*

### Friends

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

### 5.26.1 Detailed Description

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

### 5.26.2 Constructor & Destructor Documentation

#### 5.26.2.1 Flywheel()

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

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

##### Parameters

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

### 5.26.3 Member Function Documentation

#### 5.26.3.1 get\_motors()

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

Returns the motors

##### Returns

the motors used to run the flywheel

#### 5.26.3.2 get\_target()

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

Return the `target_rpm` that the flywheel is currently trying to achieve

##### Returns

`target_rpm` the target rpm

Return the current value that the `target_rpm` should be set to

### 5.26.3.3 getRPM()

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

return the velocity of the flywheel

### 5.26.3.4 is\_on\_target()

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

check if the feedback controller thinks the flywheel is on target

#### Returns

true if on target

### 5.26.3.5 Page()

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

Creates a page displaying info about the flywheel.

#### Returns

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

### 5.26.3.6 spin\_manual()

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

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

#### Parameters

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

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

#### Parameters

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



### 5.26.3.7 spin\_rpm()

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

starts or sets the target\_rpm thread at new value what control scheme is dependent on control\_style

#### Parameters

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

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

#### Parameters

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

### 5.26.3.8 SpinRpmCmd()

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

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

#### Parameters

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

#### Returns

an auto command to add to a command controller

### 5.26.3.9 stop()

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

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

stop the RPM thread and the wheel

### 5.26.3.10 WaitUntilUpToSpeedCmd()

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

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

#### Returns

an auto command to add to a command controller

## 5.26.4 Friends And Related Symbol Documentation

### 5.26.4.1 spinRPMTask

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

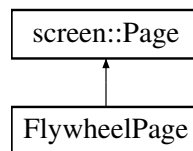
Runs a thread that keeps track of updating flywheel RPM and controlling it accordingly

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

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

## 5.27 FlywheelPage Class Reference

Inheritance diagram for FlywheelPage:



### Public Member Functions

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

### Static Public Attributes

- static const size\_t **window\_size** = 40

## 5.27.1 Member Function Documentation

### 5.27.1.1 draw()

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

See also

[Page::draw](#)

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

### 5.27.1.2 update()

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

#### See also

Page::update

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

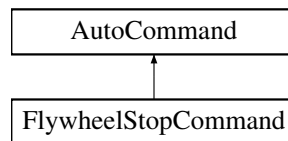
The documentation for this class was generated from the following file:

- src/subsystems/flywheel.cpp

## 5.28 FlywheelStopCommand Class Reference

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

Inheritance diagram for FlywheelStopCommand:



#### Public Member Functions

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

#### Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

#### Additional Inherited Members

#### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default\_timeout** = 10.0

### 5.28.1 Detailed Description

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

### 5.28.2 Constructor & Destructor Documentation

#### 5.28.2.1 FlywheelStopCommand()

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

Construct a [FlywheelStopCommand](#)

##### Parameters

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

### 5.28.3 Member Function Documentation

#### 5.28.3.1 run()

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

Run stop Overrides run from [AutoCommand](#)

##### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

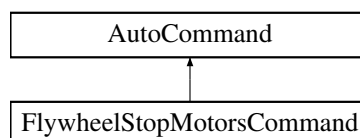
The documentation for this class was generated from the following files:

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

## 5.29 FlywheelStopMotorsCommand Class Reference

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

Inheritance diagram for FlywheelStopMotorsCommand:



## Public Member Functions

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

## Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

## Additional Inherited Members

## Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.29.1 Detailed Description

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

### 5.29.2 Constructor & Destructor Documentation

#### 5.29.2.1 FlywheelStopMotorsCommand()

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

Construct a FlywheelStopMotors Command

#### Parameters

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

### 5.29.3 Member Function Documentation

#### 5.29.3.1 run()

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

Run stop Overrides run from [AutoCommand](#)

**Returns**

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

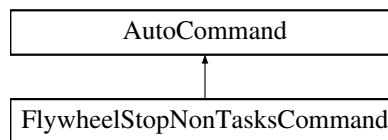
The documentation for this class was generated from the following files:

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

## 5.30 FlywheelStopNonTasksCommand Class Reference

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

Inheritance diagram for FlywheelStopNonTasksCommand:

**Additional Inherited Members****Public Member Functions inherited from [AutoCommand](#)**

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* **withTimeout** (double t\_seconds)
- [AutoCommand](#) \* **withCancelCondition** ([Condition](#) \*true\_to\_end)

**Public Attributes inherited from [AutoCommand](#)**

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* **true\_to\_end** = nullptr

**Static Public Attributes inherited from [AutoCommand](#)**

- static constexpr double **default\_timeout** = 10.0

### 5.30.1 Detailed Description

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

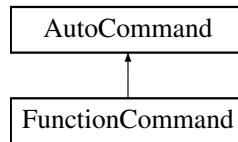
The documentation for this class was generated from the following files:

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

## 5.31 FunctionCommand Class Reference

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

Inheritance diagram for FunctionCommand:



### Public Member Functions

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

### Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* **withTimeout** (double t\_seconds)
- [AutoCommand](#) \* **withCancelCondition** ([Condition](#) \*true\_to\_end)

### Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default\_timeout** = 10.0

#### 5.31.1 Detailed Description

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

## 5.31.2 Member Function Documentation

### 5.31.2.1 run()

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

Executes the command Overridden by child classes

#### Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

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

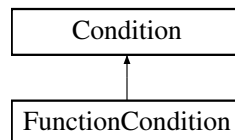
- include/utils/command\_structure/auto\_command.h

## 5.32 FunctionCondition Class Reference

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

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

Inheritance diagram for FunctionCondition:



### Public Member Functions

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

### Public Member Functions inherited from [Condition](#)

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

### 5.32.1 Detailed Description

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



## 5.32.2 Member Function Documentation

### 5.32.2.1 test()

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

Implements [Condition](#).

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

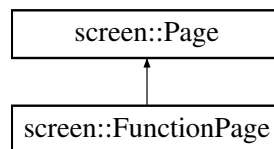
- include/utls/command\_structure/auto\_command.h
- src/utls/command\_structure/auto\_command.cpp

## 5.33 screen::FunctionPage Class Reference

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

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

Inheritance diagram for screen::FunctionPage:



### Public Member Functions

- [FunctionPage](#) (update\_func\_t update\_f, draw\_func\_t draw\_t)  
*Creates a function page.*
- void [update](#) (bool was\_pressed, int x, int y) override  
*update uses the supplied update function to update this page*
- void [draw](#) (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number) override  
*draw uses the supplied draw function to draw to the screen*

### 5.33.1 Detailed Description

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

## 5.33.2 Constructor & Destructor Documentation

### 5.33.2.1 FunctionPage()

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

Creates a function page.

[FunctionPage](#).

## Parameters

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

### 5.33.3 Member Function Documentation

#### 5.33.3.1 draw()

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

draw uses the supplied draw function to draw to the screen

See also

[Page::draw](#)

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

#### 5.33.3.2 update()

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

update uses the supplied update function to update this page

See also

[Page::update](#)

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

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

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

## 5.34 GenericAuto Class Reference

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

## Public Member Functions

- bool [run](#) (bool blocking)
- void [add](#) (state\_ptr new\_state)
- void [add\\_async](#) (state\_ptr async\_state)
- void [add\\_delay](#) (int ms)

### 5.34.1 Detailed Description

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

### 5.34.2 Member Function Documentation

#### 5.34.2.1 [add\(\)](#)

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

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

##### Parameters

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

#### 5.34.2.2 [add\\_async\(\)](#)

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

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

##### Parameters

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

#### 5.34.2.3 [add\\_delay\(\)](#)

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

[add\\_delay](#) adds a period where the auto system will simply wait for the specified time

##### Parameters

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

#### 5.34.2.4 run()

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

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

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

##### Parameters

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

##### Returns

true after all states have finished.

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

- include/utlis/generic\_auto.h
- src/utlis/generic\_auto.cpp

## 5.35 GraphDrawer Class Reference

### Public Member Functions

- [GraphDrawer](#) (int num\_samples, double lower\_bound, double upper\_bound, std::vector< vex::color > colors, size\_t num\_series=1)  
*Creates a graph drawer with the specified number of series (each series is a separate line)*
- void [add\\_samples](#) (std::vector< [point\\_t](#) > sample)
- void [add\\_samples](#) (std::vector< double > sample)
- void [draw](#) (vex::brain::lcd &screen, int x, int y, int width, int height)

### 5.35.1 Constructor & Destructor Documentation

#### 5.35.1.1 GraphDrawer()

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

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

## Parameters

<i>num_samples</i>	the number of samples to graph at a time (40 will graph the last 40 data points)
<i>lower_bound</i>	the bottom of the window when displaying (if 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

## 5.35.2 Member Function Documentation

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

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

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

## Parameters

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

### 5.35.2.2 add\_samples() [2/2]

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

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

## Parameters

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

### 5.35.2.3 draw()

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

draws the graph to the screen in the constructor

## Parameters

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

## Parameters

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

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

- include/utils/graph\_drawer.h
- src/utils/graph\_drawer.cpp

## 5.36 PurePursuit::hermite\_point Struct Reference

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

## Public Member Functions

- [point\\_t](#) **getPoint** () const
- [Vector2D](#) **getTangent** () const

## Public Attributes

- double **x**
- double **y**
- double **dir**
- double **mag**

### 5.36.1 Detailed Description

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

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

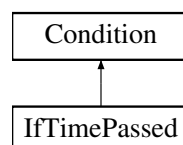
- include/utils/pure\_pursuit.h

## 5.37 IfTimePassed Class Reference

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

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

Inheritance diagram for IfTimePassed:



### Public Member Functions

- **IfTimePassed** (double time\_s)
- bool **test** () override

### Public Member Functions inherited from [Condition](#)

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

#### 5.37.1 Detailed Description

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

#### 5.37.2 Member Function Documentation

##### 5.37.2.1 test()

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

Implements [Condition](#).

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

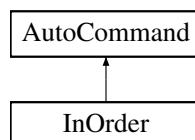
- include/utils/command\_structure/auto\_command.h
- src/utils/command\_structure/auto\_command.cpp

## 5.38 InOrder Class Reference

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

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

Inheritance diagram for InOrder:



### Public Member Functions

- **InOrder** (const [InOrder](#) &other)=default
- **InOrder** (std::queue< [AutoCommand](#) \* > cmds)
- **InOrder** (std::initializer\_list< [AutoCommand](#) \* > cmds)
- bool **run** () override
- void **on\_timeout** () override

## Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand](#) \* **withTimeout** (double t\_seconds)
- [AutoCommand](#) \* **withCancelCondition** ([Condition](#) \*true\_to\_end)

## Additional Inherited Members

## Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* **true\_to\_end** = nullptr

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default\_timeout** = 10.0

### 5.38.1 Detailed Description

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

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

### 5.38.2 Member Function Documentation

#### 5.38.2.1 on\_timeout()

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

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

Reimplemented from [AutoCommand](#).

#### 5.38.2.2 run()

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

Executes the command Overridden by child classes

#### Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

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

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



## 5.39 screen::LabelConfig Struct Reference

### Public Attributes

- std::string **label**

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

- include/subsystems/screen.h

## 5.40 Lift< T > Class Template Reference

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

### Classes

- struct [lift\\_cfg\\_t](#)

### Public Member Functions

- [Lift](#) (motor\_group &lift\_motors, [lift\\_cfg\\_t](#) &lift\_cfg, map< T, double > &setpoint\_map, limit \*homing\_↔ switch=NULL)
- void [control\\_continuous](#) (bool up\_ctrl, bool down\_ctrl)
- void [control\\_manual](#) (bool up\_btn, bool down\_btn, int volt\_up, int volt\_down)
- void [control\\_setpoints](#) (bool up\_step, bool down\_step, vector< T > pos\_list)
- bool [set\\_position](#) (T pos)
- bool [set\\_setpoint](#) (double val)
- double [get\\_setpoint](#) ()
- void [hold](#) ()
- void [home](#) ()
- bool [get\\_async](#) ()
- void [set\\_async](#) (bool val)
- void [set\\_sensor\\_function](#) (double(\*fn\_ptr)(void))
- void [set\\_sensor\\_reset](#) (void(\*fn\_ptr)(void))

### 5.40.1 Detailed Description

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

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

#### Author

Ryan McGee

## 5.40.2 Constructor & Destructor Documentation

### 5.40.2.1 Lift()

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

Construct the [Lift](#) object and begin the background task that controls the lift.

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

#### Parameters

<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

## 5.40.3 Member Function Documentation

### 5.40.3.1 control\_continuous()

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

Control the lift with an "up" button and a "down" button. Use [PID](#) to hold the lift when letting go.

#### Parameters

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

### 5.40.3.2 control\_manual()

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

Control the lift with manual controls (no holding voltage)

## Parameters

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

**5.40.3.3 control\_setpoints()**

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

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

## Parameters

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

**5.40.3.4 get\_async()**

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

## Returns

whether or not the background thread is running the lift

**5.40.3.5 get\_setpoint()**

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

## Returns

The current setpoint for the lift

**5.40.3.6 hold()**

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

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

### 5.40.3.7 home()

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

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

### 5.40.3.8 set\_async()

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

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

#### Parameters

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

### 5.40.3.9 set\_position()

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

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

#### Parameters

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

#### Returns

True if the pid has reached the setpoint

### 5.40.3.10 set\_sensor\_function()

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

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

#### Parameters

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

### 5.40.3.11 `set_sensor_reset()`

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

Creates a custom hook to reset the sensor used in `set_sensor_function()`. Example: `/code{.cpp} my_lift.set_↵ sensor_reset( my_sensor.resetPosition );/endcode`

### 5.40.3.12 `set_setpoint()`

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

Manually set a setpoint value for the lift `PID` to go to.

#### Parameters

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

#### Returns

True if the pid has reached the setpoint

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

- `include/subsystems/lift.h`

## 5.41 `Lift< T >::lift_cfg_t` Struct Reference

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

### Public Attributes

- double `up_speed`
- double `down_speed`
- double `softstop_up`
- double `softstop_down`
- `PID::pid_config_t` `lift_pid_cfg`

### 5.41.1 Detailed Description

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

`lift_cfg_t` holds the physical parameter specifications of a lify system. includes:

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

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

- `include/subsystems/lift.h`

## 5.42 Logger Class Reference

Class to simplify writing to files.

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

### Public Member Functions

- [Logger](#) (const std::string &filename)  
*Create a logger that will save to a file.*
- [Logger](#) (const [Logger](#) &l)=delete  
*copying not allowed*
- [Logger](#) & [operator=](#) (const [Logger](#) &l)=delete  
*copying not allowed*
- void [Log](#) (const std::string &s)  
*Write a string to the log.*
- void [Log](#) (LogLevel level, const std::string &s)  
*Write a string to the log with a loglevel.*
- void [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*

### 5.42.1 Detailed Description

Class to simplify writing to files.

### 5.42.2 Constructor & Destructor Documentation

#### 5.42.2.1 Logger()

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

Create a logger that will save to a file.

## Parameters

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

## 5.42.3 Member Function Documentation

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

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

Write a string to the log.

## Parameters

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

### 5.42.3.2 Log() [2/2]

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

Write a string to the log with a loglevel.

## Parameters

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

### 5.42.3.3 Logf() [1/2]

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

Write a formatted string to the log.

## Parameters

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

**5.42.3.4 Logf() [2/2]**

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

Write a formatted string to the log with a loglevel.

**Parameters**

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

**5.42.3.5 Logln() [1/2]**

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

Write a string and newline to the log.

**Parameters**

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

**5.42.3.6 Logln() [2/2]**

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

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

**Parameters**

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

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

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

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

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



### Public Attributes

- double **max\_v**  
*the maximum velocity the robot can drive*
- double **accel**  
*the most acceleration the robot can do*
- [PID::pid\\_config\\_t](#) **pid\_cfg**  
*configuration parameters for the internal [PID](#) controller*
- [FeedForward::ff\\_config\\_t](#) **ff\_cfg**  
*configuration parameters for the internal*

### 5.43.1 Detailed Description

m\_profile\_config holds all data the motion controller uses to plan paths When motion profile is given a target to drive to, max\_v and accel are used to make the trapezoid profile instructing the controller how to drive pid\_cfg, ff\_cfg are used to find the motor outputs necessary to execute this path

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

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

## 5.44 Mat2 Struct Reference

### Public Member Functions

- [point\\_t](#) **operator\*** (const [point\\_t](#) p) const

### Static Public Member Functions

- static [Mat2](#) **FromRotationDegrees** (double degrees)

### Public Attributes

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

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

- include/utils/geometry.h

## 5.45 MecanumDrive Class Reference

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

## Classes

- struct [mecanumdrive\\_config\\_t](#)

## Public Member Functions

- [MecanumDrive](#) (vex::motor &left\_front, vex::motor &right\_front, vex::motor &left\_rear, vex::motor &right\_rear, vex::rotation \*lateral\_wheel=NULL, vex::inertial \*imu=NULL, [mecanumdrive\\_config\\_t](#) \*config=NULL)
- void [drive\\_raw](#) (double direction\_deg, double magnitude, double rotation)
- void [drive](#) (double left\_y, double left\_x, double right\_x, int power=2)
- bool [auto\\_drive](#) (double inches, double direction, double speed, bool gyro\_correction=true)
- bool [auto\\_turn](#) (double degrees, double speed, bool ignore\_imu=false)

### 5.45.1 Detailed Description

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

### 5.45.2 Constructor & Destructor Documentation

#### 5.45.2.1 MecanumDrive()

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

Create the Mecanum drivetrain object

### 5.45.3 Member Function Documentation

#### 5.45.3.1 auto\_drive()

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

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

#### Parameters

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

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

#### Parameters

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

#### Returns

Whether or not the maneuver is complete.

#### 5.45.3.2 auto\_turn()

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

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

#### Parameters

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

#### Returns

whether or not the robot has finished the maneuver

### 5.45.3.3 drive()

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

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

#### Parameters

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

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

#### Parameters

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

### 5.45.3.4 drive\_raw()

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

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

#### Parameters

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

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

- include/subsystems/mecanum\_drive.h

- `src/subsystems/mecanum_drive.cpp`

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

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

### Public Attributes

- [PID::pid\\_config\\_t](#) `drive_pid_conf`
- [PID::pid\\_config\\_t](#) `drive_gyro_pid_conf`
- [PID::pid\\_config\\_t](#) `turn_pid_conf`
- double `drive_wheel_diam`
- double `lateral_wheel_diam`
- double `wheelbase_width`

### 5.46.1 Detailed Description

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

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

- `include/subsystems/mecanum_drive.h`

## 5.47 motion\_t Struct Reference

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

### Public Attributes

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

### 5.47.1 Detailed Description

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

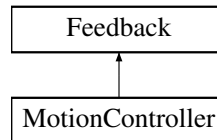
The documentation for this struct was generated from the following file:

- `include/utils/controls/trapezoid_profile.h`

## 5.48 MotionController Class Reference

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

Inheritance diagram for MotionController:



### Classes

- struct [m\\_profile\\_cfg\\_t](#)

### Public Member Functions

- [MotionController](#) ([m\\_profile\\_cfg\\_t](#) &config)  
*Construct a new Motion Controller object.*
- void [init](#) (double start\_pt, double end\_pt, double start\_vel, double end\_vel) override  
*Initialize the motion profile for a new movement This will also reset the [PID](#) and profile timers.*
- double [update](#) (double sensor\_val) override  
*Update the motion profile with a new sensor value.*
- double [get](#) () override
- void [set\\_limits](#) (double lower, double upper) override
- bool [is\\_on\\_target](#) () override
- [motion\\_t get\\_motion](#) () const
- [screen::Page](#) \* [Page](#) ()

### Static Public Member Functions

- static [FeedForward::ff\\_config\\_t tune\\_feedforward](#) ([TankDrive](#) &drive, [OdometryTank](#) &odometry, double pct=0.6, double duration=2)

### Friends

- class [MotionControllerPage](#)

### 5.48.1 Detailed Description

Motion Controller class

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

This takes the constants kS, kV, kA, kP, kI, kD, max\_v and acceleration and wraps around a feedforward, [PID](#) and trapezoid profile. It does so with the following formula:

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

For [PID](#) and Feedforward specific formulae, see [pid.h](#), [feedforward.h](#), and [trapezoid\\_profile.h](#)

Author

Ryan McGee

Date

7/13/2022

## 5.48.2 Constructor & Destructor Documentation

### 5.48.2.1 MotionController()

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

Construct a new Motion Controller object.

#### Parameters

<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
---------------	--

## 5.48.3 Member Function Documentation

### 5.48.3.1 get()

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

#### Returns

the last saved result from the feedback controller

Implements [Feedback](#).

### 5.48.3.2 get\_motion()

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

#### Returns

The current position, velocity and acceleration setpoints

### 5.48.3.3 init()

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

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

#### Parameters

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

Implements [Feedback](#).

#### 5.48.3.4 is\_on\_target()

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

##### Returns

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

Implements [Feedback](#).

#### 5.48.3.5 set\_limits()

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

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

##### Parameters

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

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

##### Parameters

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

Implements [Feedback](#).

#### 5.48.3.6 tune\_feedforward()

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

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

Next is kV (voltage to sustain a certain velocity), where the robot will record 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  $kA*Accel = pct-kV*V-kS$ , the reciprocal of the linear regression is the kA value.



## Parameters

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

## Returns

A tuned feedforward object

## 5.48.3.7 update()

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

Update the motion profile with a new sensor value.

## Parameters

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

## Returns

the motor input generated from the motion profile

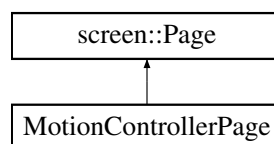
Implements [Feedback](#).

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

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

## 5.49 MotionControllerPage Class Reference

Inheritance diagram for MotionControllerPage:



## Public Member Functions

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

### 5.49.1 Member Function Documentation

#### 5.49.1.1 draw()

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

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

##### Parameters

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

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

#### 5.49.1.2 update()

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

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

##### Parameters

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

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

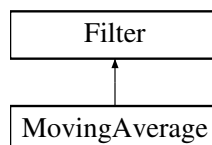
The documentation for this class was generated from the following file:

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

## 5.50 MovingAverage Class Reference

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

Inheritance diagram for MovingAverage:



## Public Member Functions

- [MovingAverage](#) (int buffer\_size)
- [MovingAverage](#) (int buffer\_size, double starting\_value)
- void [add\\_entry](#) (double n) override
- double [get\\_value](#) () const override
- int [get\\_size](#) () const

### 5.50.1 Detailed Description

#### [MovingAverage](#)

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

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

WARNING: because we need a lot of samples to get the actual value, the value given by the [MovingAverage](#) will 'lag' behind the actual value that the sensor is reading. Using a [MovingAverage](#) is thus a tradeoff between accuracy and lag time (more samples) vs. less accuracy and faster updating (less samples).

### 5.50.2 Constructor & Destructor Documentation

#### 5.50.2.1 [MovingAverage\(\)](#) [1/2]

```
MovingAverage::MovingAverage (
    int buffer_size )
```

Create a moving average calculator with 0 as the default value

##### Parameters

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

#### 5.50.2.2 [MovingAverage\(\)](#) [2/2]

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

Create a moving average calculator with a specified default value

##### Parameters

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

### 5.50.3 Member Function Documentation

#### 5.50.3.1 add\_entry()

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

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

##### Parameters

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

Implements [Filter](#).

#### 5.50.3.2 get\_size()

```
int MovingAverage::get_size ( ) const
```

How many samples the average is made from

##### Returns

the number of samples used to calculate this average

#### 5.50.3.3 get\_value()

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

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

##### Returns

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

How many samples the average is made from

##### Returns

the number of samples used to calculate this average

Implements [Filter](#).

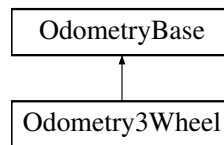
The documentation for this class was generated from the following files:

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

## 5.51 Odometry3Wheel Class Reference

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

Inheritance diagram for Odometry3Wheel:



### Classes

- struct [odometry3wheel\\_cfg\\_t](#)

### Public Member Functions

- [Odometry3Wheel](#) ([CustomEncoder](#) &lside\_fwd, [CustomEncoder](#) &rside\_fwd, [CustomEncoder](#) &off\_axis, [odometry3wheel\\_cfg\\_t](#) &cfg, bool is\_async=true)
- [pose\\_t update](#) () override
- void [tune](#) (vex::controller &con, [TankDrive](#) &drive)

### Public Member Functions inherited from [OdometryBase](#)

- [OdometryBase](#) (bool is\_async)
- [pose\\_t get\\_position](#) (void)
- virtual void [set\\_position](#) (const [pose\\_t](#) &newpos=[zero\\_pos](#))
- [AutoCommand](#) \* [SetPositionCmd](#) (const [pose\\_t](#) &newpos=[zero\\_pos](#))
- void [end\\_async](#) ()
- double [get\\_speed](#) ()
- double [get\\_accel](#) ()
- double [get\\_angular\\_speed\\_deg](#) ()
- double [get\\_angular\\_accel\\_deg](#) ()

### Additional Inherited Members

### Static Public Member Functions inherited from [OdometryBase](#)

- static int [background\\_task](#) (void \*ptr)
- static double [pos\\_diff](#) ([pose\\_t](#) start\_pos, [pose\\_t](#) end\_pos)
- static double [rot\\_diff](#) ([pose\\_t](#) pos1, [pose\\_t](#) pos2)
- static double [smallest\\_angle](#) (double start\_deg, double end\_deg)

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

- bool [end\\_task](#) = false  
*end\_task is true if we instruct the odometry thread to shut down*

## Static Public Attributes inherited from [OdometryBase](#)

- static constexpr [pose\\_t](#) [zero\\_pos](#) = {.x=0.0L, .y=0.0L, .rot=90.0L}

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

- vex::task \* [handle](#)
- vex::mutex [mut](#)
- [pose\\_t](#) [current\\_pos](#)
- double [speed](#)
- double [accel](#)
- double [ang\\_speed\\_deg](#)
- double [ang\\_accel\\_deg](#)

### 5.51.1 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 it's movement in the background.

#### Author

Ryan McGee

#### Date

Oct 31 2022

### 5.51.2 Constructor & Destructor Documentation

#### 5.51.2.1 [Odometry3Wheel\(\)](#)

```
Odometry3Wheel::Odometry3Wheel (
    CustomEncoder & lside_fwd,
    CustomEncoder & rside_fwd,
    CustomEncoder & off_axis,
    odometry3wheel_cfg_t & cfg,
    bool is_async = true )
```

Construct a new Odometry 3 Wheel object

**Parameters**

<i>lside_fwd</i>	left-side encoder reference
<i>rside_fwd</i>	right-side encoder reference
<i>off_axis</i>	off-axis (perpendicular) encoder reference
<i>cfg</i>	robot odometry configuration
<i>is_async</i>	true to constantly run in the background

**5.51.3 Member Function Documentation****5.51.3.1 tune()**

```
void Odometry3Wheel::tune (
    vex::controller & con,
    TankDrive & drive )
```

A guided tuning process to automatically find tuning parameters. This method is blocking, and returns when tuning has finished. Follow the instructions on the controller to complete the tuning process

**Parameters**

<i>con</i>	Controller reference, for screen and button control
<i>drive</i>	Drivetrain reference for robot control

A guided tuning process to automatically find tuning parameters. This method is blocking, and returns when tuning has finished. Follow the instructions on the controller to complete the tuning process

It is assumed the gear ratio and encoder PPR have been set correctly

**5.51.3.2 update()**

```
pose_t Odometry3Wheel::update ( ) [override], [virtual]
```

Update the current position of the robot once, using the current state of the encoders and the previous known location

**Returns**

the robot's updated position

Implements [OdometryBase](#).

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

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

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

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

### Public Attributes

- double [wheelbase\\_dist](#)
- double [off\\_axis\\_center\\_dist](#)
- double [wheel\\_diam](#)

### 5.52.1 Detailed Description

[odometry3wheel\\_cfg\\_t](#) holds all the specifications for how to calculate position with 3 encoders See the core wiki for what exactly each of these parameters measures

### 5.52.2 Member Data Documentation

#### 5.52.2.1 off\_axis\_center\_dist

```
double Odometry3Wheel::odometry3wheel_cfg_t::off_axis_center_dist
```

distance from the center of the robot to the center off axis wheel

#### 5.52.2.2 wheel\_diam

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

the diameter of the tracking wheel

#### 5.52.2.3 wheelbase\_dist

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

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

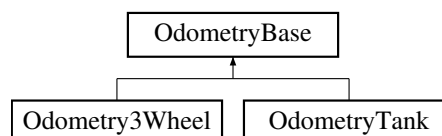
The documentation for this struct was generated from the following file:

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

## 5.53 OdometryBase Class Reference

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

Inheritance diagram for OdometryBase:





### Public Member Functions

- [OdometryBase](#) (bool is\_async)
- [pose\\_t get\\_position](#) (void)
- virtual void [set\\_position](#) (const [pose\\_t](#) &newpos=[zero\\_pos](#))
- [AutoCommand](#) \* [SetPositionCmd](#) (const [pose\\_t](#) &newpos=[zero\\_pos](#))
- virtual [pose\\_t update](#) ()=0
- void [end\\_async](#) ()
- double [get\\_speed](#) ()
- double [get\\_accel](#) ()
- double [get\\_angular\\_speed\\_deg](#) ()
- double [get\\_angular\\_accel\\_deg](#) ()

### Static Public Member Functions

- static int [background\\_task](#) (void \*ptr)
- static double [pos\\_diff](#) ([pose\\_t](#) start\_pos, [pose\\_t](#) end\_pos)
- static double [rot\\_diff](#) ([pose\\_t](#) pos1, [pose\\_t](#) pos2)
- static double [smallest\\_angle](#) (double start\_deg, double end\_deg)

### Public Attributes

- bool [end\\_task](#) = false  
*end\_task is true if we instruct the odometry thread to shut down*

### Static Public Attributes

- static constexpr [pose\\_t zero\\_pos](#) = {.x=0.0L, .y=0.0L, .rot=90.0L}

### Protected Attributes

- vex::task \* [handle](#)
- vex::mutex [mut](#)
- [pose\\_t current\\_pos](#)
- double [speed](#)
- double [accel](#)
- double [ang\\_speed\\_deg](#)
- double [ang\\_accel\\_deg](#)

## 5.53.1 Detailed Description

### [OdometryBase](#)

This base class contains all the shared code between different implementations of odometry. It handles the asynchronous management, position input/output and basic math functions, and holds positional types specific to field orientation.

All future odometry implementations should extend this file and redefine [update\(\)](#) function.

#### Author

Ryan McGee

#### Date

Aug 11 2021

## 5.53.2 Constructor & Destructor Documentation

### 5.53.2.1 OdometryBase()

```
OdometryBase::OdometryBase (
    bool is_async )
```

Construct a new Odometry Base object

#### Parameters

<i>is_async</i>	True to run constantly in the background, false to call <a href="#">update()</a> manually
-----------------	---

## 5.53.3 Member Function Documentation

### 5.53.3.1 background\_task()

```
int OdometryBase::background_task (
    void * ptr ) [static]
```

Function that runs in the background task. This function pointer is passed to the `vex::task` constructor.

#### Parameters

<i>ptr</i>	Pointer to <a href="#">OdometryBase</a> object
------------	--

#### Returns

Required integer return code. Unused.

### 5.53.3.2 end\_async()

```
void OdometryBase::end_async ( )
```

End the background task. Cannot be restarted. If the user wants to end the thread but keep the data up to date, they must run the [update\(\)](#) function manually from then on.

### 5.53.3.3 get\_accel()

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

Get the current acceleration

#### Returns

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

#### 5.53.3.4 get\_angular\_accel\_deg()

```
double OdometryBase::get_angular_accel_deg ( )
```

Get the current angular acceleration in degrees

##### Returns

the angular acceleration at which we are turning (deg/s<sup>2</sup>)

#### 5.53.3.5 get\_angular\_speed\_deg()

```
double OdometryBase::get_angular_speed_deg ( )
```

Get the current angular speed in degrees

##### Returns

the angular velocity at which we are turning (deg/s)

#### 5.53.3.6 get\_position()

```
pose_t OdometryBase::get_position (
    void )
```

Gets the current position and rotation

##### Returns

the position that the odometry believes the robot is at

Gets the current position and rotation

#### 5.53.3.7 get\_speed()

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

Get the current speed

##### Returns

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

#### 5.53.3.8 pos\_diff()

```
double OdometryBase::pos_diff (
    pose_t start_pos,
    pose_t end_pos ) [static]
```

Get the distance between two points

## Parameters

<i>start_pos</i>	distance from this point
<i>end_pos</i>	to this point

## Returns

the euclidean distance between *start\_pos* and *end\_pos*

**5.53.3.9 rot\_diff()**

```
double OdometryBase::rot_diff (
    pose_t pos1,
    pose_t pos2 ) [static]
```

Get the change in rotation between two points

## Parameters

<i>pos1</i>	position with initial rotation
<i>pos2</i>	position with final rotation

## Returns

change in rotation between *pos1* and *pos2*

Get the change in rotation between two points

**5.53.3.10 set\_position()**

```
void OdometryBase::set_position (
    const pose_t & newpos = zero_pos ) [virtual]
```

Sets the current position of the robot

## Parameters

<i>newpos</i>	the new position that the odometry will believe it is at
---------------	--

Sets the current position of the robot

Reimplemented in [OdometryTank](#).

**5.53.3.11 smallest\_angle()**

```
double OdometryBase::smallest_angle (
    double start_deg,
    double end_deg ) [static]
```

Get the smallest difference in angle between a start heading and end heading. Returns the difference between -180 degrees and +180 degrees, representing the robot turning left or right, respectively.

#### Parameters

<i>start_deg</i>	intitial angle (degrees)
<i>end_deg</i>	final angle (degrees)

#### Returns

the smallest angle from the initial to the final angle. This takes into account the wrapping of rotations around 360 degrees

Get the smallest difference in angle between a start heading and end heading. Returns the difference between -180 degrees and +180 degrees, representing the robot turning left or right, respectively.

#### 5.53.3.12 update()

```
virtual pose_t OdometryBase::update ( ) [pure virtual]
```

Update the current position on the field based on the sensors

#### Returns

the location that the robot is at after the odometry does its calculations

Implemented in [Odometry3Wheel](#), and [OdometryTank](#).

### 5.53.4 Member Data Documentation

#### 5.53.4.1 accel

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

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

#### 5.53.4.2 ang\_accel\_deg

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

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

#### 5.53.4.3 ang\_speed\_deg

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

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

#### 5.53.4.4 current\_pos

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

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

#### 5.53.4.5 handle

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

handle to the vex task that is running the odometry code

#### 5.53.4.6 mut

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

Mutex to control multithreading

#### 5.53.4.7 speed

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

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

#### 5.53.4.8 zero\_pos

```
constexpr pose_t OdometryBase::zero_pos = {.x=0.0L, .y=0.0L, .rot=90.0L} [inline], [static], [constexpr]
```

Zeroed position. X=0, Y=0, Rotation= 90 degrees

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

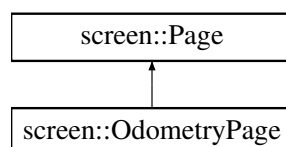
- include/subsystems/odometry/odometry\_base.h
- src/subsystems/odometry/odometry\_base.cpp

## 5.54 screen::OdometryPage Class Reference

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

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

Inheritance diagram for screen::OdometryPage:



## Public Member Functions

- [OdometryPage](#) ([OdometryBase](#) &odom, double robot\_width, double robot\_height, bool do\_trail)  
*Create an odometry trail. Make sure odometry is initlized before now.*
- void [update](#) (bool was\_pressed, int x, int y) override
- void [draw](#) (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number) override

### 5.54.1 Detailed Description

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

### 5.54.2 Constructor & Destructor Documentation

#### 5.54.2.1 OdometryPage()

```
screen::OdometryPage::OdometryPage (
    OdometryBase & odom,
    double robot_width,
    double robot_height,
    bool do_trail )
```

Create an odometry trail. Make sure odometry is initlized before now.

#### Parameters

<i>odom</i>	the odometry system to monitor
<i>robot_width</i>	the width (side to side) of the robot in inches. Used for visualization
<i>robot_height</i>	the robot_height (front to back) of the robot in inches. Used for visualization
<i>do_trail</i>	whether or not to calculate and draw the trail. Drawing and storing takes a very <i>slight</i> extra amount of processing power

### 5.54.3 Member Function Documentation

#### 5.54.3.1 draw()

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

#### See also

[Page::draw](#)

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

### 5.54.3.2 update()

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

See also

[Page::update](#)

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

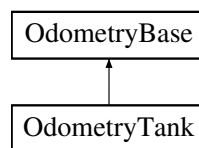
The documentation for this class was generated from the following files:

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

## 5.55 OdometryTank Class Reference

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

Inheritance diagram for OdometryTank:



### Public Member Functions

- [OdometryTank](#) (vex::motor\_group &left\_side, vex::motor\_group &right\_side, [robot\\_specs\\_t](#) &config, vex::inertial \*imu=NULL, bool is\_async=true)
- [OdometryTank](#) ([CustomEncoder](#) &left\_custom\_enc, [CustomEncoder](#) &right\_custom\_enc, [robot\\_specs\\_t](#) &config, vex::inertial \*imu=NULL, bool is\_async=true)
- [OdometryTank](#) (vex::encoder &left\_vex\_enc, vex::encoder &right\_vex\_enc, [robot\\_specs\\_t](#) &config, vex::inertial \*imu=NULL, bool is\_async=true)
- [pose\\_t update](#) () override
- void [set\\_position](#) (const [pose\\_t](#) &newpos=[zero\\_pos](#)) override

### Public Member Functions inherited from [OdometryBase](#)

- [OdometryBase](#) (bool is\_async)
- [pose\\_t get\\_position](#) (void)
- [AutoCommand](#) \* [SetPositionCmd](#) (const [pose\\_t](#) &newpos=[zero\\_pos](#))
- void [end\\_async](#) ()
- double [get\\_speed](#) ()
- double [get\\_accel](#) ()
- double [get\\_angular\\_speed\\_deg](#) ()
- double [get\\_angular\\_accel\\_deg](#) ()



## Additional Inherited Members

### Static Public Member Functions inherited from [OdometryBase](#)

- static int [background\\_task](#) (void \*ptr)
- static double [pos\\_diff](#) ([pose\\_t](#) start\_pos, [pose\\_t](#) end\_pos)
- static double [rot\\_diff](#) ([pose\\_t](#) pos1, [pose\\_t](#) pos2)
- static double [smallest\\_angle](#) (double start\_deg, double end\_deg)

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

- bool [end\\_task](#) = false  
*end\_task is true if we instruct the odometry thread to shut down*

### Static Public Attributes inherited from [OdometryBase](#)

- static constexpr [pose\\_t](#) [zero\\_pos](#) = {.x=0.0L, .y=0.0L, .rot=90.0L}

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

- vex::task \* [handle](#)
- vex::mutex [mut](#)
- [pose\\_t](#) [current\\_pos](#)
- double [speed](#)
- double [accel](#)
- double [ang\\_speed\\_deg](#)
- double [ang\\_accel\\_deg](#)

## 5.55.1 Detailed Description

[OdometryTank](#) defines an odometry system for a tank drivetrain. This requires encoders in the same orientation as the drive wheels. Odometry is a "start and forget" subsystem, which means once it's created and configured, it will constantly run in the background and track the robot's X, Y and rotation coordinates.

## 5.55.2 Constructor & Destructor Documentation

### 5.55.2.1 [OdometryTank\(\)](#) [1/3]

```
OdometryTank::OdometryTank (
    vex::motor_group & left_side,
    vex::motor_group & right_side,
    robot\_specs\_t & config,
    vex::inertial * imu = NULL,
    bool is_async = true )
```

Initialize the Odometry module, calculating position from the drive motors.

## Parameters

<i>left_side</i>	The left motors
<i>right_side</i>	The right motors
<i>config</i>	the specifications that supply the odometry with descriptions of the robot. See <a href="#">robot_specs_t</a> for what is contained
<i>imu</i>	The robot's inertial sensor. If not included, rotation is calculated from the encoders.
<i>is_async</i>	If true, position will be updated in the background continuously. If false, the programmer will have to manually call <a href="#">update()</a> .

## 5.55.2.2 OdometryTank() [2/3]

```
OdometryTank::OdometryTank (
    CustomEncoder & left_custom_enc,
    CustomEncoder & right_custom_enc,
    robot_specs_t & config,
    vex::inertial * imu = NULL,
    bool is_async = true )
```

Initialize the Odometry module, calculating position from the drive motors.

## Parameters

<i>left_custom_enc</i>	The left custom encoder
<i>right_custom_enc</i>	The right custom encoder
<i>config</i>	the specifications that supply the odometry with descriptions of the robot. See <a href="#">robot_specs_t</a> for what is contained
<i>imu</i>	The robot's inertial sensor. If not included, rotation is calculated from the encoders.
<i>is_async</i>	If true, position will be updated in the background continuously. If false, the programmer will have to manually call <a href="#">update()</a> .

## 5.55.2.3 OdometryTank() [3/3]

```
OdometryTank::OdometryTank (
    vex::encoder & left_vex_enc,
    vex::encoder & right_vex_enc,
    robot_specs_t & config,
    vex::inertial * imu = NULL,
    bool is_async = true )
```

Initialize the Odometry module, calculating position from the drive motors.

## Parameters

<i>left_vex_enc</i>	The left vex encoder
<i>right_vex_enc</i>	The right vex encoder
<i>config</i>	the specifications that supply the odometry with descriptions of the robot. See <a href="#">robot_specs_t</a> for what is contained
<i>imu</i>	The robot's inertial sensor. If not included, rotation is calculated from the encoders.
<i>is_async</i>	If true, position will be updated in the background continuously. If false, the programmer will have to manually call <a href="#">update()</a> .

### 5.55.3 Member Function Documentation

#### 5.55.3.1 `set_position()`

```
void OdometryTank::set_position (
    const pose\_t & newpos = zero\_pos ) [override], [virtual]
```

`set_position` tells the odometry to place itself at a position

##### Parameters

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

Resets the position and rotational data to the input.

Reimplemented from [OdometryBase](#).

#### 5.55.3.2 `update()`

```
pose\_t OdometryTank::update ( ) [override], [virtual]
```

Update the current position on the field based on the sensors

##### Returns

the position that odometry has calculated itself to be at

Update, store and return the current position of the robot. Only use if not initializing with a separate thread.

Implements [OdometryBase](#).

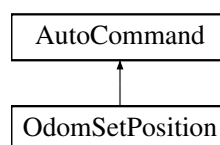
The documentation for this class was generated from the following files:

- `include/subsystems/odometry/odometry_tank.h`
- `src/subsystems/odometry/odometry_tank.cpp`

## 5.56 OdomSetPosition Class Reference

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

Inheritance diagram for OdomSetPosition:



## Public Member Functions

- [OdomSetPosition](#) ([OdometryBase](#) &odom, const [pose\\_t](#) &newpos=[OdometryBase::zero\\_pos](#))
- bool [run](#) () override

## Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

## Additional Inherited Members

## Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.56.1 Detailed Description

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

### 5.56.2 Constructor & Destructor Documentation

#### 5.56.2.1 OdomSetPosition()

```
OdomSetPosition::OdomSetPosition (
    OdometryBase & odom,
    const pose\_t & newpos = OdometryBase::zero\_pos )
```

constructs a new [OdomSetPosition](#) command

#### Parameters

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

Construct an Odometry set pos

#### Parameters

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

### 5.56.3 Member Function Documentation

#### 5.56.3.1 run()

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

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

##### Returns

true when execution is complete, false otherwise

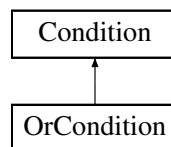
Reimplemented from [AutoCommand](#).

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

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

## 5.57 OrCondition Class Reference

Inheritance diagram for OrCondition:



### Public Member Functions

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

### Public Member Functions inherited from [Condition](#)

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

### 5.57.1 Member Function Documentation

#### 5.57.1.1 test()

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

Implements [Condition](#).

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

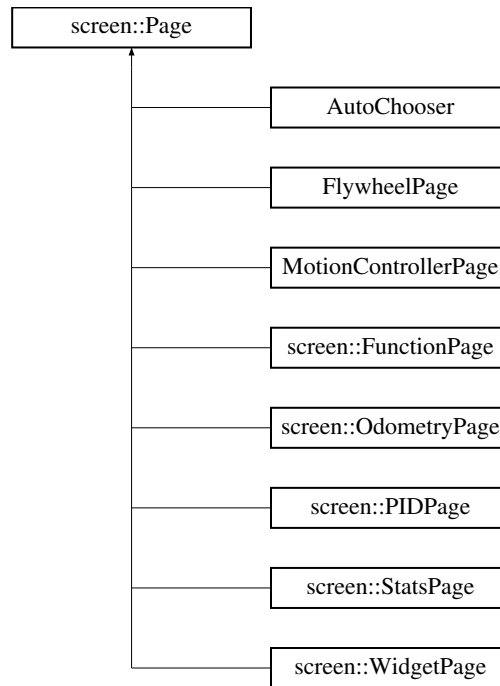
- src/utls/command\_structure/auto\_command.cpp

## 5.58 screen::Page Class Reference

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

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

Inheritance diagram for screen::Page:



### Public Member Functions

- virtual void [update](#) (bool was\_pressed, int x, int y)  
*collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this [Page](#) (only drawn page gets touch updates))*
- virtual void [draw](#) (vex::brain::lcd &screen, bool first\_draw, unsigned int frame\_number)  
*draw stored data to the screen (runs at 10 hz and only runs if this page is in front)*

### 5.58.1 Detailed Description

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

### 5.58.2 Member Function Documentation

#### 5.58.2.1 draw()

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

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

## Parameters

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

Reimplemented in [AutoChooser](#), [screen::WidgetPage](#), [screen::StatsPage](#), [screen::OdometryPage](#), [screen::FunctionPage](#), [screen::PIDPage](#), [MotionControllerPage](#), and [FlywheelPage](#).

**5.58.2.2 update()**

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

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

## Parameters

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

Reimplemented in [AutoChooser](#), [screen::WidgetPage](#), [screen::StatsPage](#), [screen::OdometryPage](#), [screen::FunctionPage](#), [screen::PIDPage](#), [MotionControllerPage](#), and [FlywheelPage](#).

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

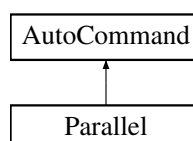
- include/subsystems/screen.h

**5.59 Parallel Class Reference**

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

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

Inheritance diagram for [Parallel](#):



**Public Member Functions**

- **Parallel** (std::initializer\_list< [AutoCommand](#) \* > cmds)
- bool [run](#) () override
- void [on\\_timeout](#) () override

**Public Member Functions inherited from [AutoCommand](#)**

- [AutoCommand](#) \* **withTimeout** (double t\_seconds)
- [AutoCommand](#) \* **withCancelCondition** ([Condition](#) \*true\_to\_end)

**Additional Inherited Members****Public Attributes inherited from [AutoCommand](#)**

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

**Static Public Attributes inherited from [AutoCommand](#)**

- static constexpr double **default\_timeout** = 10.0

**5.59.1 Detailed Description**

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

**5.59.2 Member Function Documentation****5.59.2.1 [on\\_timeout\(\)](#)**

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

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

Reimplemented from [AutoCommand](#).

**5.59.2.2 [run\(\)](#)**

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

Executes the command Overridden by child classes

**Returns**

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

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

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



## 5.60 parallel\_runner\_info Struct Reference

### Public Attributes

- int **index**
- std::vector< vex::task \* > \* **runners**
- [AutoCommand](#) \* **cmd**

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

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

## 5.61 PurePursuit::Path Class Reference

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

### Public Member Functions

- [Path](#) (std::vector< [point\\_t](#) > points, double radius)
- std::vector< [point\\_t](#) > [get\\_points](#) ()
- double [get\\_radius](#) ()
- bool [is\\_valid](#) ()

### 5.61.1 Detailed Description

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

### 5.61.2 Constructor & Destructor Documentation

#### 5.61.2.1 Path()

```
PurePursuit::Path::Path (
    std::vector< point\_t > points,
    double radius )
```

Create a [Path](#)

#### Parameters

<i>points</i>	the points that make up the path
<i>radius</i>	the lookahead radius for pure pursuit

### 5.61.3 Member Function Documentation

#### 5.61.3.1 get\_points()

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

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

#### 5.61.3.2 get\_radius()

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

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

#### 5.61.3.3 is\_valid()

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

Get whether this path will behave as expected

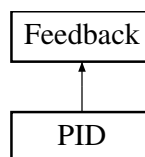
The documentation for this class was generated from the following files:

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

## 5.62 PID Class Reference

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

Inheritance diagram for PID:



### Classes

- struct [pid\\_config\\_t](#)

### Public Types

- enum [ERROR\\_TYPE](#) { **LINEAR** , **ANGULAR** }

## Public Member Functions

- [PID](#) ([pid\\_config\\_t](#) &[config](#))
- void [init](#) (double start\_pt, double set\_pt, double start\_vel=0, double end\_vel=0) override
- double [update](#) (double sensor\_val) override
- double [get\\_sensor\\_val](#) () const  
*gets the sensor value that we were last updated with*
- double [get](#) () override
- void [set\\_limits](#) (double lower, double upper) override
- bool [is\\_on\\_target](#) () override
- void [reset](#) ()
- double [get\\_error](#) ()
- double [get\\_target](#) () const
- void [set\\_target](#) (double target)

## Public Attributes

- [pid\\_config\\_t](#) & [config](#)

### 5.62.1 Detailed Description

#### [PID](#) Class

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

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

The [PID](#) object will determine it is "on target" when the error is within the deadband, for a duration of on\_target\_time

#### Author

Ryan McGee

#### Date

4/3/2020

### 5.62.2 Member Enumeration Documentation

#### 5.62.2.1 [ERROR\\_TYPE](#)

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

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

### 5.62.3 Constructor & Destructor Documentation

#### 5.62.3.1 [PID\(\)](#)

```
PID::PID (
    pid\_config\_t & config )
```

Create the [PID](#) object

## Parameters

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

Create the [PID](#) object

## 5.62.4 Member Function Documentation

### 5.62.4.1 `get()`

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

Gets the current [PID](#) out value, from when [update\(\)](#) was last run

## Returns

the Out value of the controller (voltage, RPM, whatever the [PID](#) controller is controlling)

Gets the current [PID](#) out value, from when [update\(\)](#) was last run

Implements [Feedback](#).

### 5.62.4.2 `get_error()`

```
double PID::get_error ( )
```

Get the delta between the current sensor data and the target

## Returns

the error calculated. how it is calculated depends on `error_method` specified in [pid\\_config\\_t](#)

Get the delta between the current sensor data and the target

### 5.62.4.3 `get_sensor_val()`

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

gets the sensor value that we were last updated with

## Returns

`sensor_val`

#### 5.62.4.4 `get_target()`

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

Get the [PID](#)'s target

##### Returns

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

#### 5.62.4.5 `init()`

```
void PID::init (
    double start_pt,
    double set_pt,
    double start_vel = 0,
    double end_vel = 0 ) [override], [virtual]
```

Inherited from [Feedback](#) for interoperability. Update the setpoint and reset integral accumulation

`start_pt` can be safely ignored in this feedback controller

##### Parameters

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

Implements [Feedback](#).

#### 5.62.4.6 `is_on_target()`

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

Checks if the [PID](#) controller is on target.

##### Returns

true if the loop is within `[deadband]` for `[on_target_time]` seconds

Returns true if the loop is within `[deadband]` for `[on_target_time]` seconds

Implements [Feedback](#).

#### 5.62.4.7 `reset()`

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

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

#### 5.62.4.8 set\_limits()

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

Set the limits on the [PID](#) out. The [PID](#) out will "clip" itself to be between the limits.

##### Parameters

<i>lower</i>	the lower limit. the <a href="#">PID</a> controller will never command the output go below <i>lower</i>
<i>upper</i>	the upper limit. the <a href="#">PID</a> controller will never command the output go higher than <i>upper</i>

Set the limits on the [PID](#) out. The [PID](#) out will "clip" itself to be between the limits.

Implements [Feedback](#).

#### 5.62.4.9 set\_target()

```
void PID::set_target (
    double target )
```

Set the target for the [PID](#) loop, where the robot is trying to end up

##### Parameters

<i>target</i>	the sensor reading we would like to achieve
---------------	---

Set the target for the [PID](#) loop, where the robot is trying to end up

#### 5.62.4.10 update()

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

Update the [PID](#) loop by taking the time difference from last update, and running the [PID](#) formula with the new sensor data

##### Parameters

<i>sensor_val</i>	the distance, angle, encoder position or whatever it is we are measuring
-------------------	--

##### Returns

the new output. What would be returned by [PID::get\(\)](#)

Implements [Feedback](#).

## 5.62.5 Member Data Documentation

### 5.62.5.1 config

`pid_config_t& PID::config`

configuration struct for this controller. see [pid\\_config\\_t](#) for information about what this contains

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

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

## 5.63 PID::pid\_config\_t Struct Reference

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

### Public Attributes

- double **p**  
*proportional coeffecient  $p * error()$*
- double **i**  
*integral coeffecient  $i * integral(error)$*
- double **d**  
*derivitave coeffecient  $d * derivative(error)$*
- double **deadband**  
*at what threshold are we close enough to be finished*
- double [on\\_target\\_time](#)
- [ERROR\\_TYPE](#) [error\\_method](#)

### 5.63.1 Detailed Description

[pid\\_config\\_t](#) holds the configuration parameters for a pid controller In addition to the constant of proportional, integral and derivative, these parameters include:

- deadband -
- on\_target\_time - for how long do we have to be at the target to stop As well, [pid\\_config\\_t](#) holds an error type which determines whether errors should be calculated as if the sensor position is a measure of distance or an angle

## 5.63.2 Member Data Documentation

### 5.63.2.1 error\_method

[ERROR\\_TYPE](#) `PID::pid_config_t::error_method`

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

### 5.63.2.2 on\_target\_time

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

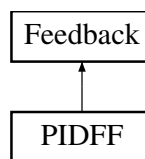
the time in seconds that we have to be on target for to say we are officially at the target

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

- include/utils/controls/pid.h

## 5.64 PIDFF Class Reference

Inheritance diagram for PIDFF:



### Public Member Functions

- **PIDFF** ([PID::pid\\_config\\_t](#) &pid\_cfg, [FeedForward::ff\\_config\\_t](#) &ff\_cfg)
- void **init** (double start\_pt, double set\_pt, double start\_vel, double end\_vel) override
- void **set\_target** (double set\_pt)
- double **get\_target** () const
- double **get\_sensor\_val** () const
- double **update** (double val) override
- double **update** (double val, double vel\_setpt, double a\_setpt=0)
- double **get** () override
- void **set\_limits** (double lower, double upper) override
- bool **is\_on\_target** () override
- void **reset** ()

### Public Attributes

- [PID](#) pid

## 5.64.1 Member Function Documentation

### 5.64.1.1 get()

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

#### Returns

the last saved result from the feedback controller

Implements [Feedback](#).



### 5.64.1.2 init()

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

Initialize the feedback controller for a movement

## Parameters

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

Initialize the feedback controller for a movement

## Parameters

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

Implements [Feedback](#).

### 5.64.1.3 is\_on\_target()

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

## Returns

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

### 5.64.1.4 set\_limits()

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

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

## Parameters

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

Implements [Feedback](#).

### 5.64.1.5 set\_target()

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

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

## Parameters

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

**5.64.1.6 update() [1/2]**

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

Iterate the feedback loop once with an updated sensor value. Only kS for feedforward will be applied.

## Parameters

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

## Returns

feedback loop result

Implements [Feedback](#).

**5.64.1.7 update() [2/2]**

```
double PIDFF::update (
    double val,
    double vel_setpt,
    double a_setpt = 0 )
```

Iterate the feedback loop once with an updated sensor value

## Parameters

<i>val</i>	value from the sensor
<i>vel_setpt</i>	Velocity for feedforward
<i>a_setpt</i>	Acceleration for feedforward

## Returns

feedback loop result

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

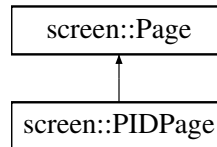
- include/utils/controls/pidff.h
- src/utils/controls/pidff.cpp

## 5.65 screen::PIDPage Class Reference

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

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

Inheritance diagram for screen::PIDPage:



### Public Member Functions

- [PIDPage](#) ([PID](#) &pid, std::string name, std::function< void(void)> onchange=[])() {}  
Create a [PIDPage](#).
- **PIDPage** ([PIDFF](#) &pidff, std::string name, std::function< void(void)> onchange=[])() {}
- void [update](#) (bool was\_pressed, int x, int y) override
- void [draw](#) (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number) override

### 5.65.1 Detailed Description

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

### 5.65.2 Constructor & Destructor Documentation

#### 5.65.2.1 PIDPage()

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

Create a [PIDPage](#).

#### Parameters

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

### 5.65.3 Member Function Documentation

#### 5.65.3.1 draw()

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

See also

[Page::draw](#)

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

#### 5.65.3.2 update()

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

See also

[Page::update](#)

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

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

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

## 5.66 point\_t Struct Reference

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

### Public Member Functions

- double [dist](#) (const [point\\_t](#) other) const
- [point\\_t operator+](#) (const [point\\_t](#) &other) const
- [point\\_t operator-](#) (const [point\\_t](#) &other) const
- [point\\_t operator\\*](#) (double s) const
- [point\\_t operator/](#) (double s) const
- [point\\_t operator-](#) () const
- [point\\_t operator+](#) () const
- bool [operator==](#) (const [point\\_t](#) &rhs)

## Public Attributes

- double **x**  
*the x position in space*
- double **y**  
*the y position in space*

### 5.66.1 Detailed Description

Data structure representing an X,Y coordinate

### 5.66.2 Member Function Documentation

#### 5.66.2.1 dist()

```
double point_t::dist (
    const point_t other ) const [inline]
```

dist calculates the euclidian distance between this point and another point using the pythagorean theorem

#### Parameters

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

#### Returns

the euclidian distance between this and other

#### 5.66.2.2 operator+()

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

[Vector2D](#) addition operation on points

#### Parameters

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

#### Returns

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

#### 5.66.2.3 operator-()

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

[Vector2D](#) subtraction operation on points

#### Parameters

<i>other</i>	the <a href="#">point_t</a> to subtract from this
--------------	---

#### Returns

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

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

- include/utils/geometry.h

## 5.67 `pose_t` Struct Reference

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

#### Public Member Functions

- [point\\_t](#) `get_point ()`

#### Public Attributes

- double **x**  
*x position in the world*
- double **y**  
*y position in the world*
- double **rot**  
*rotation in the world*

### 5.67.1 Detailed Description

Describes a single position and rotation

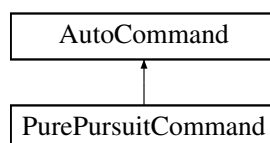
The documentation for this struct was generated from the following file:

- include/utils/geometry.h

## 5.68 `PurePursuitCommand` Class Reference

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

Inheritance diagram for `PurePursuitCommand`:



## Public Member Functions

- [PurePursuitCommand](#) ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, [PurePursuit::Path](#) path, directionType dir, double max\_speed=1, double end\_speed=0)
- bool [run](#) () override
- void [on\\_timeout](#) () override

## Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

## Additional Inherited Members

## Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.68.1 Detailed Description

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

### 5.68.2 Constructor & Destructor Documentation

#### 5.68.2.1 PurePursuitCommand()

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

Construct a Pure Pursuit [AutoCommand](#)

#### Parameters

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



### 5.68.3 Member Function Documentation

#### 5.68.3.1 on\_timeout()

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

Reset the drive system when it times out

Reimplemented from [AutoCommand](#).

#### 5.68.3.2 run()

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

Direct call to [TankDrive::pure\\_pursuit](#)

Reimplemented from [AutoCommand](#).

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

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

## 5.69 Rect Struct Reference

### Public Member Functions

- [point\\_t dimensions](#) () const
- [point\\_t center](#) () const
- double **width** () const
- double **height** () const
- bool **contains** ([point\\_t](#) p) const

### Static Public Member Functions

- static [Rect](#) **from\_min\_and\_size** ([point\\_t](#) min, [point\\_t](#) size)

### Public Attributes

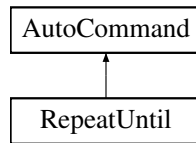
- [point\\_t](#) min
- [point\\_t](#) max

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

- include/utls/geometry.h

## 5.70 RepeatUntil Class Reference

Inheritance diagram for RepeatUntil:



### Public Member Functions

- [RepeatUntil](#) ([InOrder](#) cmds, size\_t repeats)  
*RepeatUntil that runs a fixed number of times.*
- [RepeatUntil](#) ([InOrder](#) cmds, [Condition](#) \*true\_to\_end)  
*RepeatUntil the condition.*
- bool [run](#) () override
- void [on\\_timeout](#) () override

### Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

### Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

## 5.70.1 Constructor & Destructor Documentation

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

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

[RepeatUntil](#) that runs a fixed number of times.

## Parameters

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

**5.70.1.2 RepeatUntil()** [2/2]

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

[RepeatUntil](#) the condition.

## Parameters

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

**5.70.2 Member Function Documentation****5.70.2.1 on\_timeout()**

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

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

Reimplemented from [AutoCommand](#).

**5.70.2.2 run()**

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

Executes the command Overridden by child classes

## Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

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

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

## 5.71 robot\_specs\_t Struct Reference

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

### Public Attributes

- double **robot\_radius**  
*if you were to draw a circle with this radius, the robot would be entirely contained within it*
- double **odom\_wheel\_diam**  
*the diameter of the wheels used for*
- double **odom\_gear\_ratio**  
*the ratio of the odometry wheel to the encoder reading odometry data*
- double **dist\_between\_wheels**  
*the distance between centers of the central drive wheels*
- double **drive\_correction\_cutoff**  
*the distance at which to stop trying to turn towards the target. If we are less than this value, we can continue driving forward to minimize our distance but will not try to spin around to point directly at the target*
- [Feedback](#) \* **drive\_feedback**  
*the default feedback for autonomous driving*
- [Feedback](#) \* **turn\_feedback**  
*the default feedback for autonomous turning*
- [PID::pid\\_config\\_t](#) **correction\_pid**  
*the pid controller to keep the robot driving in as straight a line as possible*

### 5.71.1 Detailed Description

Main robot characterization struct. This will be passed to all the major subsystems that require info about the robot. All distance measurements are in inches.

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

- include/robot\_specs.h

## 5.72 screen::ScreenData Struct Reference

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

### Public Member Functions

- **ScreenData** (const std::vector< [Page](#) \* > &m\_pages, int m\_page, vex::brain::lcd &m\_screen)

### Public Attributes

- std::vector< [Page](#) \* > **pages**
- int **page** = 0
- vex::brain::lcd **screen**

### 5.72.1 Detailed Description

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

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

- `src/subsystems/screen.cpp`

## 5.73 screen::ScreenRect Struct Reference

### Public Attributes

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

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

- `include/subsystems/screen.h`

## 5.74 Serializer Class Reference

Serializes Arbitrary data to a file on the SD Card.

```
#include <serializer.h>
```

### Public Member Functions

- `~Serializer ()`  
*Save and close upon destruction (bc of vex, this doesnt always get called when the program ends. To be sure, call `save_to_disk`)*
- `Serializer (const std::string &filename, bool flush_always=true)`  
*create a [Serializer](#)*
- `void save_to_disk () const`  
*saves current [Serializer](#) state to disk*
- `void set_int (const std::string &name, int i)`  
*Setters - not saved until `save_to_disk` is called.*
- `void set_bool (const std::string &name, bool b)`  
*sets a bool by the name of name to b. If `flush_always == true`, this will save to the sd card*
- `void set_double (const std::string &name, double d)`  
*sets a double by the name of name to d. If `flush_always == true`, this will save to the sd card*
- `void set_string (const std::string &name, std::string str)`  
*sets a string by the name of name to s. If `flush_always == true`, this will save to the sd card*
- `int int_or (const std::string &name, int otherwise)`  
*gets a value stored in the serializer. If not found, sets the value to otherwise*
- `bool bool_or (const std::string &name, bool otherwise)`  
*gets a value stored in the serializer. If not, sets the value to otherwise*
- `double double_or (const std::string &name, double otherwise)`  
*gets a value stored in the serializer. If not, sets the value to otherwise*
- `std::string string_or (const std::string &name, std::string otherwise)`  
*gets a value stored in the serializer. If not, sets the value to otherwise*

### 5.74.1 Detailed Description

Serializes Arbitrary data to a file on the SD Card.

### 5.74.2 Constructor & Destructor Documentation

#### 5.74.2.1 Serializer()

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

create a [Serializer](#)

##### Parameters

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

### 5.74.3 Member Function Documentation

#### 5.74.3.1 bool\_or()

```
bool Serializer::bool_or (
    const std::string & name,
    bool otherwise )
```

gets a value stored in the serializer. If not, sets the value to otherwise

##### Parameters

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

##### Returns

the value if found or otherwise

#### 5.74.3.2 double\_or()

```
double Serializer::double_or (
    const std::string & name,
    double otherwise )
```

gets a value stored in the serializer. If not, sets the value to otherwise

**Parameters**

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

**Returns**

the value if found or otherwise

**5.74.3.3 int\_or()**

```
int Serializer::int_or (
    const std::string & name,
    int otherwise )
```

gets a value stored in the serializer. If not found, sets the value to otherwise

Getters Return value if it exists in the serializer

**Parameters**

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

**Returns**

the value if found or otherwise

**5.74.3.4 save\_to\_disk()**

```
void Serializer::save_to_disk ( ) const
```

saves current [Serializer](#) state to disk

forms data bytes then saves to filename this was opened with

**5.74.3.5 set\_bool()**

```
void Serializer::set_bool (
    const std::string & name,
    bool b )
```

sets a bool by the name of name to b. If flush\_always == true, this will save to the sd card

**Parameters**

<i>name</i>	name of bool
<i>b</i>	value of bool

### 5.74.3.6 set\_double()

```
void Serializer::set_double (
    const std::string & name,
    double d )
```

sets a double by the name of name to d. If flush\_always == true, this will save to the sd card

#### Parameters

<i>name</i>	name of double
<i>d</i>	value of double

### 5.74.3.7 set\_int()

```
void Serializer::set_int (
    const std::string & name,
    int i )
```

Setters - not saved until save\_to\_disk is called.

sets an integer by the name of name to i. If flush\_always == true, this will save to the sd card

#### Parameters

<i>name</i>	name of integer
<i>i</i>	value of integer

### 5.74.3.8 set\_string()

```
void Serializer::set_string (
    const std::string & name,
    std::string str )
```

sets a string by the name of name to s. If flush\_always == true, this will save to the sd card

#### Parameters

<i>name</i>	name of string
<i>i</i>	value of string

### 5.74.3.9 string\_or()

```
std::string Serializer::string_or (
    const std::string & name,
    std::string otherwise )
```

gets a value stored in the serializer. If not, sets the value to otherwise



**Parameters**

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

**Returns**

the value if found or otherwise

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

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

## 5.75 screen::SizedWidget Struct Reference

**Public Attributes**

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

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

- include/subsystems/screen.h

## 5.76 SliderCfg Struct Reference

**Public Attributes**

- double & **val**
- double **min**
- double **max**

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

- include/subsystems/layout.h

## 5.77 screen::SliderConfig Struct Reference

**Public Attributes**

- double & **val**
- double **low**
- double **high**

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

- include/subsystems/screen.h

## 5.78 screen::SliderWidget Class Reference

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

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

### Public Member Functions

- [SliderWidget](#) (double &val, double low, double high, [Rect](#) rect, std::string name)  
*Creates a slider widget.*
- bool [update](#) (bool was\_pressed, int x, int y)  
*responds to user input*
- void [draw](#) (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number)  
*Page::draws the slide to the screen*

### 5.78.1 Detailed Description

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

### 5.78.2 Constructor & Destructor Documentation

#### 5.78.2.1 SliderWidget()

```
screen::SliderWidget::SliderWidget (
    double & val,
    double low,
    double high,
    Rect rect,
    std::string name ) [inline]
```

Creates a slider widget.

#### Parameters

<i>val</i>	reference to the value to modify
<i>low</i>	minimum value to go to
<i>high</i>	maximum value to go to
<i>rect</i>	rect to draw it
<i>name</i>	name of the value

### 5.78.3 Member Function Documentation

#### 5.78.3.1 update()

```
bool screen::SliderWidget::update (
```

```

    bool was_pressed,
    int x,
    int y )

```

responds to user input

#### Parameters

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

#### Returns

true if the value updated

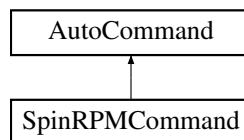
The documentation for this class was generated from the following files:

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

## 5.79 SpinRPMCommand Class Reference

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

Inheritance diagram for SpinRPMCommand:



#### Public Member Functions

- [SpinRPMCommand](#) ([Flywheel](#) &flywheel, int rpm)
- bool [run](#) () override

#### Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

#### Additional Inherited Members

#### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default\_timeout** = 10.0

### 5.79.1 Detailed Description

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

### 5.79.2 Constructor & Destructor Documentation

#### 5.79.2.1 SpinRPMCommand()

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

Construct a SpinRPM Command

##### Parameters

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

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

### 5.79.3 Member Function Documentation

#### 5.79.3.1 run()

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

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

##### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

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

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

## 5.80 PurePursuit::spline Struct Reference

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

### Public Member Functions

- double **getY** (double x)

### Public Attributes

- double **a**
- double **b**
- double **c**
- double **d**
- double **x\_start**
- double **x\_end**

### 5.80.1 Detailed Description

Represents a piece of a cubic spline with  $s(x) = a(x-x_i)^3 + b(x-x_i)^2 + c(x-x_i) + d$  The `x_start` and `x_end` shows where the equation is valid.

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

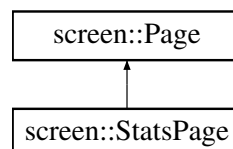
- include/utils/pure\_pursuit.h

## 5.81 screen::StatsPage Class Reference

Draws motor stats and battery stats to the screen.

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

Inheritance diagram for `screen::StatsPage`:



### Public Member Functions

- [StatsPage](#) (std::map< std::string, vex::motor & > motors)  
*Creates a stats page.*
- void [update](#) (bool was\_pressed, int x, int y) override
- void [draw](#) (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number) override

### 5.81.1 Detailed Description

Draws motor stats and battery stats to the screen.

### 5.81.2 Constructor & Destructor Documentation

#### 5.81.2.1 StatsPage()

```
screen::StatsPage::StatsPage (
    std::map< std::string, vex::motor & > motors )
```

Creates a stats page.

Parameters

<i>motors</i>	a map of string to motor that we want to draw on this page
---------------	--

### 5.81.3 Member Function Documentation

#### 5.81.3.1 draw()

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

See also

[Page::draw](#)

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

#### 5.81.3.2 update()

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

See also

[Page::update](#)

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

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

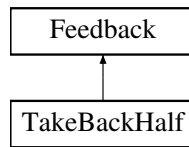
- include/subsystems/screen.h
- src/subsystems/screen.cpp

## 5.82 TakeBackHalf Class Reference

A velocity controller.

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

Inheritance diagram for TakeBackHalf:



### Public Member Functions

- **TakeBackHalf** (double [TBH\\_gain](#), double first\_cross\_split, double on\_target\_threshold)
- void [init](#) (double start\_pt, double set\_pt, double, double)
- double [update](#) (double val) override
- double [get](#) () override
- void [set\\_limits](#) (double lower, double upper) override
- bool [is\\_on\\_target](#) () override

### Public Attributes

- double **TBH\_gain**  
*tuned parameter*
- double **first\_cross\_split**

### 5.82.1 Detailed Description

A velocity controller.

#### Warning

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

### 5.82.2 Member Function Documentation

#### 5.82.2.1 [get\(\)](#)

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

#### Returns

the last saved result from the feedback controller

Implements [Feedback](#).

#### 5.82.2.2 [init\(\)](#)

```
void TakeBackHalf::init (
    double start_pt,
    double set_pt,
    double ,
    double ) [virtual]
```

Initialize the feedback controller for a movement

## Parameters

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

Implements [Feedback](#).

**5.82.2.3 is\_on\_target()**

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

## Returns

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

**5.82.2.4 set\_limits()**

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

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

## Parameters

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

Implements [Feedback](#).

**5.82.2.5 update()**

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

Iterate the feedback loop once with an updated sensor value

## Parameters

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



**Returns**

feedback loop result

Implements [Feedback](#).

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

- include/utils/controls/take\_back\_half.h
- src/utils/controls/take\_back\_half.cpp

## 5.83 TankDrive Class Reference

```
#include <tank_drive.h>
```

**Public Types**

- enum class [BrakeType](#) { [None](#) , [ZeroVelocity](#) , [Smart](#) }

**Public Member Functions**

- [TankDrive](#) (motor\_group &left\_motors, motor\_group &right\_motors, [robot\\_specs\\_t](#) &config, [OdometryBase](#) \*odom=NULL)
- [AutoCommand](#) \* [DriveToPointCmd](#) ([point\\_t](#) pt, vex::directionType dir=vex::forward, double max\_speed=1.0, double end\_speed=0.0)
- [AutoCommand](#) \* [DriveToPointCmd](#) ([Feedback](#) &fb, [point\\_t](#) pt, vex::directionType dir=vex::forward, double max\_speed=1.0, double end\_speed=0.0)
- [AutoCommand](#) \* [DriveForwardCmd](#) (double dist, vex::directionType dir=vex::forward, double max\_speed=1.0, double end\_speed=0.0)
- [AutoCommand](#) \* [DriveForwardCmd](#) ([Feedback](#) &fb, double dist, vex::directionType dir=vex::forward, double max\_speed=1.0, double end\_speed=0.0)
- [AutoCommand](#) \* [TurnToHeadingCmd](#) (double heading, double max\_speed=1.0, double end\_speed=0.0)
- [AutoCommand](#) \* [TurnToHeadingCmd](#) ([Feedback](#) &fb, double heading, double max\_speed=1.0, double end\_speed=0.0)
- [AutoCommand](#) \* [TurnDegreesCmd](#) (double degrees, double max\_speed=1.0, double start\_speed=0.0)
- [AutoCommand](#) \* [TurnDegreesCmd](#) ([Feedback](#) &fb, double degrees, double max\_speed=1.0, double end\_speed=0.0)
- [AutoCommand](#) \* [PurePursuitCmd](#) ([PurePursuit::Path](#) path, directionType dir, double max\_speed=1, double end\_speed=0)
- [AutoCommand](#) \* [PurePursuitCmd](#) ([Feedback](#) &feedback, [PurePursuit::Path](#) path, directionType dir, double max\_speed=1, double end\_speed=0)
- void [stop](#) ()
- void [drive\\_tank](#) (double left, double right, int power=1, [BrakeType](#) bt=[BrakeType::None](#))
- void [drive\\_tank\\_raw](#) (double left, double right)
- void [drive\\_arcade](#) (double forward\_back, double left\_right, int power=1, [BrakeType](#) bt=[BrakeType::None](#))
- bool [drive\\_forward](#) (double inches, directionType dir, [Feedback](#) &feedback, double max\_speed=1, double end\_speed=0)
- bool [drive\\_forward](#) (double inches, directionType dir, double max\_speed=1, double end\_speed=0)
- bool [turn\\_degrees](#) (double degrees, [Feedback](#) &feedback, double max\_speed=1, double end\_speed=0)
- bool [turn\\_degrees](#) (double degrees, double max\_speed=1, double end\_speed=0)

- bool [drive\\_to\\_point](#) (double x, double y, vex::directionType dir, [Feedback](#) &feedback, double max\_speed=1, double end\_speed=0)
- bool [drive\\_to\\_point](#) (double x, double y, vex::directionType dir, double max\_speed=1, double end\_speed=0)
- bool [turn\\_to\\_heading](#) (double heading\_deg, [Feedback](#) &feedback, double max\_speed=1, double end\_speed=0)
- bool [turn\\_to\\_heading](#) (double heading\_deg, double max\_speed=1, double end\_speed=0)
- void [reset\\_auto](#) ()
- bool [pure\\_pursuit](#) ([PurePursuit::Path](#) path, directionType dir, [Feedback](#) &feedback, double max\_speed=1, double end\_speed=0)
- bool [pure\\_pursuit](#) ([PurePursuit::Path](#) path, directionType dir, double max\_speed=1, double end\_speed=0)

### Static Public Member Functions

- static double [modify\\_inputs](#) (double input, int power=2)

## 5.83.1 Detailed Description

[TankDrive](#) is a class to run a tank drive system. A tank drive system, sometimes called differential drive, has a motor (or group of synchronized motors) on the left and right side

## 5.83.2 Member Enumeration Documentation

### 5.83.2.1 BrakeType

```
enum class TankDrive::BrakeType [strong]
```

#### Enumerator

None	just send 0 volts to the motors
ZeroVelocity	try to bring the robot to rest. But don't try to hold position
Smart	bring the robot to rest and once it's stopped, try to hold that position

## 5.83.3 Constructor & Destructor Documentation

### 5.83.3.1 TankDrive()

```
TankDrive::TankDrive (
    motor_group & left_motors,
    motor_group & right_motors,
    robot\_specs\_t & config,
    OdometryBase * odom = NULL )
```

Create the [TankDrive](#) object

#### Parameters

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

## 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

## 5.83.4 Member Function Documentation

### 5.83.4.1 drive\_arcade()

```
void TankDrive::drive_arcade (
    double forward_back,
    double left_right,
    int power = 1,
    BrakeType bt = BrakeType::None )
```

Drive the robot using arcade style controls. forward\_back controls the linear motion, left\_right controls the turning.

forward\_back and left\_right are in "percent": -1.0 -> 1.0

## Parameters

<i>forward_back</i>	the percent to move forward or backward
<i>left_right</i>	the percent to turn left or right
<i>power</i>	modifies the input velocities $\text{left}^{\text{power}}$ , $\text{right}^{\text{power}}$
<i>bt</i>	breaktype. What to do if the driver lets go of the sticks

Drive the robot using arcade style controls. forward\_back controls the linear motion, left\_right controls the turning.

left\_motors and right\_motors are in "percent": -1.0 -> 1.0

### 5.83.4.2 drive\_forward() [1/2]

```
bool TankDrive::drive_forward (
    double inches,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

Autonomously drive the robot forward a certain distance

## Parameters

<i>inches</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Autonomously drive the robot forward a certain distance

#### Parameters

<i>inches</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

#### Returns

true if we have finished driving to our point

#### 5.83.4.3 drive\_forward() [2/2]

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

Use odometry to drive forward a certain distance using a custom feedback controller

Returns whether or not the robot has reached it's destination.

#### Parameters

<i>inches</i>	the distance to drive forward
<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the custom feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

#### Returns

true when we have reached our target distance

Use odometry to drive forward a certain distance using a custom feedback controller

Returns whether or not the robot has reached it's destination.

#### Parameters

<i>inches</i>	the distance to drive forward
<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the custom feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

#### 5.83.4.4 drive\_tank()

```
void TankDrive::drive_tank (
    double left,
    double right,
    int power = 1,
    BrakeType bt = BrakeType::None )
```

Drive the robot using differential style controls. left\_motors controls the left motors, right\_motors controls the right motors.

left\_motors and right\_motors are in "percent": -1.0 -> 1.0

##### Parameters

<i>left</i>	the percent to run the left motors
<i>right</i>	the percent to run the right motors
<i>power</i>	modifies the input velocities $\text{left}^{\text{power}}$ , $\text{right}^{\text{power}}$
<i>bt</i>	breaktype. What to do if the driver lets go of the sticks

#### 5.83.4.5 drive\_tank\_raw()

```
void TankDrive::drive_tank_raw (
    double left,
    double right )
```

Drive the robot raw-ly

##### Parameters

<i>left</i>	the percent to run the left motors (-1, 1)
<i>right</i>	the percent to run the right motors (-1, 1)

#### 5.83.4.6 drive\_to\_point() [1/2]

```
bool TankDrive::drive_to_point (
    double x,
    double y,
    vex::directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot. Here we use the default feedback controller from the drive\_sys

Returns whether or not the robot has reached it's destination.

##### Parameters

<i>x</i>	the x position of the target
----------	------------------------------

## Parameters

<i>y</i>	the y position of the target
<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot. Here we use the default feedback controller from the drive\_sys

Returns whether or not the robot has reached it's destination.

## Parameters

<i>x</i>	the x position of the target
<i>y</i>	the y position of the target
<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

## Returns

true if we have reached our target point

## 5.83.4.7 drive\_to\_point() [2/2]

```
bool TankDrive::drive_to_point (
    double x,
    double y,
    vex::directionType dir,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0 )
```

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot.

Returns whether or not the robot has reached it's destination.

## Parameters

<i>x</i>	the x position of the target
<i>y</i>	the y position of the target
<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot.

Returns whether or not the robot has reached it's destination.

## Parameters

<i>x</i>	the x position of the target
<i>y</i>	the y position of the target
<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

## Returns

true if we have reached our target point

**5.83.4.8 modify\_inputs()**

```
double TankDrive::modify_inputs (
    double input,
    int power = 2 ) [static]
```

Create a curve for the inputs, so that drivers have more control at lower speeds. Curves are exponential, with the default being squaring the inputs.

## Parameters

<i>input</i>	the input before modification
<i>power</i>	the power to raise input to

## Returns

$\text{input}^{\text{power}}$  (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

$\text{input}^{\text{power}}$  accounting for any sign issues that would arise with this naive solution

**5.83.4.9 pure\_pursuit()** [1/2]

```
bool TankDrive::pure_pursuit (
    PurePursuit::Path path,
    directionType dir,
```

```
double max_speed = 1,
double end_speed = 0 )
```

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

Use the default drive feedback

#### Parameters

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

#### Returns

True when the path is complete

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

Use the default drive feedback

#### Parameters

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

#### Returns

True when the path is complete

#### 5.83.4.10 pure\_pursuit() [2/2]

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

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

#### Parameters

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>feedback</i>	The feedback controller determining speed
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion



**Returns**

True when the path is complete

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

**Parameters**

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

**Returns**

True when the path is complete

**5.83.4.11 reset\_auto()**

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

Reset the initialization for autonomous drive functions

**5.83.4.12 stop()**

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

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

**5.83.4.13 turn\_degrees() [1/2]**

```
bool TankDrive::turn_degrees (
    double degrees,
    double max_speed = 1,
    double end_speed = 0 )
```

Autonomously turn the robot X degrees to counterclockwise (negative for clockwise), with a maximum motor speed of percent\_speed (-1.0 -> 1.0)

Uses the default turning feedback of the drive system.

**Parameters**

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Autonomously turn the robot X degrees to counterclockwise (negative for clockwise), with a maximum motor speed of percent\_speed (-1.0 -> 1.0)

Uses the default turning feedback of the drive system.

#### Parameters

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

#### Returns

true if we turned to target number of degrees

#### 5.83.4.14 turn\_degrees() [2/2]

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

Autonomously turn the robot X degrees counterclockwise (negative for clockwise), with a maximum motor speed of percent\_speed (-1.0 -> 1.0)

Uses PID + Feedforward for it's control.

#### Parameters

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power

Autonomously turn the robot X degrees to counterclockwise (negative for clockwise), with a maximum motor speed of percent\_speed (-1.0 -> 1.0)

Uses the specified feedback for it's control.

#### Parameters

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

#### Returns

true if we have turned our target number of degrees

**5.83.4.15 turn\_to\_heading() [1/2]**

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

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

**Parameters**

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

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

**Parameters**

<i>heading_deg</i>	the heading to which we will turn
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

**Returns**

true if we have reached our target heading

**5.83.4.16 turn\_to\_heading() [2/2]**

```
bool TankDrive::turn_to_heading (
    double heading_deg,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0 )
```

Turn the robot in place to an exact heading relative to the field. 0 is forward.

**Parameters**

<i>heading_deg</i>	the heading to which we will turn
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Turn the robot in place to an exact heading relative to the field. 0 is forward.

## Parameters

<i>heading_deg</i>	the heading to which we will turn
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

## Returns

true if we have reached our target heading

The documentation for this class was generated from the following files:

- include/subsystems/tank\_drive.h
- src/subsystems/tank\_drive.cpp

## 5.84 screen::TextConfig Struct Reference

## Public Attributes

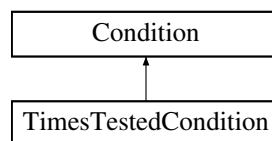
- std::function< std::string()> **text**

The documentation for this struct was generated from the following file:

- include/subsystems/screen.h

## 5.85 TimesTestedCondition Class Reference

Inheritance diagram for TimesTestedCondition:



## Public Member Functions

- **TimesTestedCondition** (size\_t N)
- bool **test** () override

Public Member Functions inherited from **Condition**

- **Condition** \* **Or** (**Condition** \*b)
- **Condition** \* **And** (**Condition** \*b)

### 5.85.1 Member Function Documentation

#### 5.85.1.1 test()

```
bool TimesTestedCondition::test ( ) [inline], [override], [virtual]
```

Implements [Condition](#).

The documentation for this class was generated from the following file:

- include/utils/command\_structure/auto\_command.h

## 5.86 trapezoid\_profile\_segment\_t Struct Reference

```
#include <trapezoid_profile.h>
```

### Public Attributes

- double **pos\_after**  
*1d position after this segment concludes*
- double **vel\_after**  
*1d velocity after this segment concludes*
- double **accel**  
*1d acceleration during the segment*
- double **duration**  
*duration of the segment*

### 5.86.1 Detailed Description

[trapezoid\\_profile\\_segment\\_t](#) is a description of one constant acceleration segment of a trapezoid motion profile

The documentation for this struct was generated from the following file:

- include/utils/controls/trapezoid\_profile.h

## 5.87 TrapezoidProfile Class Reference

```
#include <trapezoid_profile.h>
```

## Public Member Functions

- [TrapezoidProfile](#) (double max\_v, double accel)  
*Construct a new Trapezoid Profile object.*
- [motion\\_t calculate](#) (double time\_s, double pos\_s)  
*Run the trapezoidal profile based on the time and distance that's elapsed.*
- [motion\\_t calculate\\_time\\_based](#) (double time\_s)  
*Run the trapezoidal profile based on the time that's elapsed.*
- void [set\\_endpts](#) (double start, double end)  
*set\_endpts defines a start and end position*
- void [set\\_vel\\_endpts](#) (double start, double end)  
*set start and end velocities*
- void [set\\_accel](#) (double accel)  
*set\_accel sets the acceleration this profile will use (the left and right legs of the trapezoid)*
- void [set\\_max\\_v](#) (double max\_v)  
*sets the maximum velocity for the profile (the height of the top of the trapezoid)*
- double [get\\_movement\\_time](#) () const  
*uses the kinematic equations to and specified accel and max\_v to figure out how long moving along the profile would take*
- double [get\\_max\\_v](#) () const
- double [get\\_accel](#) () const

### 5.87.1 Detailed Description

#### Trapezoid Profile

This is a motion profile defined by:

- maximum acceleration
- maximum velocity
- start position and velocity
- end position and velocity

Using this information, a parametric function is generated, with a period of acceleration, constant velocity, and deceleration. The velocity graph usually looks like a trapezoid, giving it its name.

If the maximum velocity is set high enough, this will become a S-curve profile, with only acceleration and deceleration.

If the initial velocity is in the wrong direction, the profile will first come to a stop, then continue a normal trapezoid profile.

If the initial velocity is higher than the maximum velocity, the profile will first try to achieve the maximum velocity.

If the end velocity is not achievable, the profile will try to get as close as possible. The end velocity must be in the direction of the end point.

This class is designed for use in properly modelling the motion of the robots to create a feedforward and target for [PID](#). Acceleration and Maximum velocity should be measured on the robot and tuned down slightly to account for battery drop.

Here are the equations graphed for ease of understanding: <https://www.desmos.com/calculator/rkm3ivulyk>

#### Author

Ryan McGee

#### Date

7/12/2022

## 5.87.2 Constructor & Destructor Documentation

### 5.87.2.1 TrapezoidProfile()

```
TrapezoidProfile::TrapezoidProfile (
    double max_v,
    double accel )
```

Construct a new Trapezoid Profile object.

#### Parameters

<i>max_v</i>	Maximum velocity the robot can run at
<i>accel</i>	Maximum acceleration of the robot

## 5.87.3 Member Function Documentation

### 5.87.3.1 calculate()

```
motion_t TrapezoidProfile::calculate (
    double time_s,
    double pos_s )
```

Run the trapezoidal profile based on the time and distance that's elapsed.

#### Parameters

<i>time_s</i>	Time since start of movement
<i>pos_s</i>	The current position

#### Returns

[motion\\_t](#) Position, velocity and acceleration

### 5.87.3.2 calculate\_time\_based()

```
motion_t TrapezoidProfile::calculate_time_based (
    double time_s )
```

Run the trapezoidal profile based on the time that's elapsed.

#### Parameters

<i>time_s</i>	Time since start of movement
---------------	------------------------------

**Returns**

[motion\\_t](#) Position, velocity and acceleration

**5.87.3.3 get\_movement\_time()**

```
double TrapezoidProfile::get_movement_time ( ) const
```

uses the kinematic equations to and specified accel and max\_v to figure out how long moving along the profile would take

**Returns**

the time the path will take to travel

**5.87.3.4 set\_accel()**

```
void TrapezoidProfile::set_accel (
    double accel )
```

set\_accel sets the acceleration this profile will use (the left and right legs of the trapezoid)

**Parameters**

<i>accel</i>	the acceleration amount to use
--------------	--------------------------------

**5.87.3.5 set\_endpts()**

```
void TrapezoidProfile::set_endpts (
    double start,
    double end )
```

set\_endpts defines a start and end position

**Parameters**

<i>start</i>	the starting position of the path
<i>end</i>	the ending position of the path

**5.87.3.6 set\_max\_v()**

```
void TrapezoidProfile::set_max_v (
    double max_v )
```

sets the maximum velocity for the profile (the height of the top of the trapezoid)



## Parameters

<i>max</i> ↔ _v	the maximum velocity the robot can travel at
--------------------	--

**5.87.3.7 set\_vel\_endpts()**

```
void TrapezoidProfile::set_vel_endpts (
    double start,
    double end )
```

set start and end velocities

## Parameters

<i>start</i>	the starting velocity of the path
<i>end</i>	the ending velocity of the path

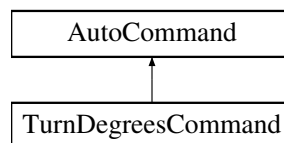
The documentation for this class was generated from the following files:

- include/utils/controls/trapezoid\_profile.h
- src/utils/trapezoid\_profile.cpp

**5.88 TurnDegreesCommand Class Reference**

```
#include <drive_commands.h>
```

Inheritance diagram for TurnDegreesCommand:

**Public Member Functions**

- [TurnDegreesCommand](#) ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, double degrees, double max\_speed=1, double end\_speed=0)
- bool [run](#) () override
- void [on\\_timeout](#) () override

**Public Member Functions inherited from [AutoCommand](#)**

- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

## Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

## 5.88.1 Detailed Description

[AutoCommand](#) wrapper class for the turn\_degrees function in the [TankDrive](#) class

## 5.88.2 Constructor & Destructor Documentation

### 5.88.2.1 TurnDegreesCommand()

```
TurnDegreesCommand::TurnDegreesCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    double degrees,
    double max_speed = 1,
    double end_speed = 0 )
```

Construct a [TurnDegreesCommand](#) Command

#### Parameters

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the turn
<i>degrees</i>	how many degrees to rotate
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

## 5.88.3 Member Function Documentation

### 5.88.3.1 on\_timeout()

```
void TurnDegreesCommand::on_timeout ( ) [override], [virtual]
```

Cleans up drive system if we time out before finishing

reset the drive system if we timeout

Reimplemented from [AutoCommand](#).

### 5.88.3.2 run()

```
bool TurnDegreesCommand::run ( ) [override], [virtual]
```

Run turn\_degrees Overrides run from [AutoCommand](#)

#### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

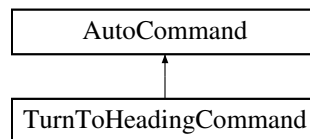
The documentation for this class was generated from the following files:

- include/utls/command\_structure/drive\_commands.h
- src/utls/command\_structure/drive\_commands.cpp

## 5.89 TurnToHeadingCommand Class Reference

```
#include <drive_commands.h>
```

Inheritance diagram for TurnToHeadingCommand:



#### Public Member Functions

- [TurnToHeadingCommand](#) ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, double heading\_deg, double speed=1, double end\_speed=0)
- bool [run](#) () override
- void [on\\_timeout](#) () override

#### Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

#### Additional Inherited Members

#### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default\_timeout** = 10.0

### 5.89.1 Detailed Description

[AutoCommand](#) wrapper class for the `turn_to_heading()` function in the [TankDrive](#) class

### 5.89.2 Constructor & Destructor Documentation

#### 5.89.2.1 TurnToHeadingCommand()

```
TurnToHeadingCommand::TurnToHeadingCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    double heading_deg,
    double max_speed = 1,
    double end_speed = 0 )
```

Construct a [TurnToHeadingCommand](#) Command

##### Parameters

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the drive
<i>heading_deg</i>	the heading to turn to in degrees
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

### 5.89.3 Member Function Documentation

#### 5.89.3.1 on\_timeout()

```
void TurnToHeadingCommand::on_timeout ( ) [override], [virtual]
```

Cleans up drive system if we time out before finishing

reset the drive system if we don't hit our target

Reimplemented from [AutoCommand](#).

#### 5.89.3.2 run()

```
bool TurnToHeadingCommand::run ( ) [override], [virtual]
```

Run `turn_to_heading` Overrides run from [AutoCommand](#)

**Returns**

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command\_structure/drive\_commands.h
- src/utils/command\_structure/drive\_commands.cpp

## 5.90 Vector2D Class Reference

```
#include <vector2d.h>
```

**Public Member Functions**

- [Vector2D](#) (double dir, double mag)
- [Vector2D](#) (point\_t p)
- double [get\\_dir](#) () const
- double [get\\_mag](#) () const
- double [get\\_x](#) () const
- double [get\\_y](#) () const
- [Vector2D](#) [normalize](#) ()
- [point\\_t](#) [point](#) ()
- [Vector2D](#) [operator\\*](#) (const double &x)
- [Vector2D](#) [operator+](#) (const [Vector2D](#) &other)
- [Vector2D](#) [operator-](#) (const [Vector2D](#) &other)

### 5.90.1 Detailed Description

[Vector2D](#) is an x,y pair Used to represent 2D locations on the field. It can also be treated as a direction and magnitude

### 5.90.2 Constructor & Destructor Documentation

#### 5.90.2.1 Vector2D() [1/2]

```
Vector2D::Vector2D (
    double dir,
    double mag )
```

Construct a vector object.

**Parameters**

<i>dir</i>	Direction, in radians. 'foward' is 0, clockwise positive when viewed from the top.
<i>mag</i>	Magnitude.

### 5.90.2.2 Vector2D() [2/2]

```
Vector2D::Vector2D (
    point_t p )
```

Construct a vector object from a cartesian point.

#### Parameters

<i>p</i>	<code>point_t.x</code> , <code>point_t.y</code>
----------	---

## 5.90.3 Member Function Documentation

### 5.90.3.1 get\_dir()

```
double Vector2D::get_dir ( ) const
```

Get the direction of the vector, in radians. '0' is forward, clockwise positive when viewed from the top.

Use `r2d()` to convert.

#### Returns

the direction of the vector in radians

Get the direction of the vector, in radians. '0' is forward, clockwise positive when viewed from the top.

Use `r2d()` to convert.

### 5.90.3.2 get\_mag()

```
double Vector2D::get_mag ( ) const
```

#### Returns

the magnitude of the vector

Get the magnitude of the vector

### 5.90.3.3 get\_x()

```
double Vector2D::get_x ( ) const
```

#### Returns

the X component of the vector; positive to the right.

Get the X component of the vector; positive to the right.

#### 5.90.3.4 `get_y()`

```
double Vector2D::get_y ( ) const
```

##### Returns

the Y component of the vector, positive forward.

Get the Y component of the vector, positive forward.

#### 5.90.3.5 `normalize()`

```
Vector2D Vector2D::normalize ( )
```

Changes the magnitude of the vector to 1

##### Returns

the normalized vector

Changes the magnetude of the vector to 1

#### 5.90.3.6 `operator*()`

```
Vector2D Vector2D::operator* (
    const double & x )
```

Scales a [Vector2D](#) by a scalar with the `*` operator

##### Parameters

<code>x</code>	the value to scale the vector by
----------------	----------------------------------

##### Returns

the this [Vector2D](#) scaled by x

#### 5.90.3.7 `operator+()`

```
Vector2D Vector2D::operator+ (
    const Vector2D & other )
```

Add the components of two vectors together [Vector2D](#) + [Vector2D](#) = (this.x + other.x, this.y + other.y)

##### Parameters

<code>other</code>	the vector to add to this
--------------------	---------------------------

**Returns**

the sum of the vectors

**5.90.3.8 operator-()**

```
Vector2D Vector2D::operator- (
    const Vector2D & other )
```

Subtract the components of two vectors together `Vector2D - Vector2D = (this.x - other.x, this.y - other.y)`

**Parameters**

<i>other</i>	the vector to subtract from this
--------------	----------------------------------

**Returns**

the difference of the vectors

**5.90.3.9 point()**

```
point_t Vector2D::point ( )
```

Returns a point from the vector

**Returns**

the point represented by the vector

Convert a direction and magnitude representation to an x, y representation

**Returns**

the x, y representation of the vector

The documentation for this class was generated from the following files:

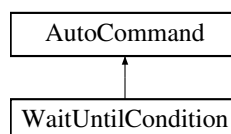
- include/utils/vector2d.h
- src/utils/vector2d.cpp

## 5.91 WaitUntilCondition Class Reference

Waits until the condition is true.

```
#include <auto_command.h>
```

Inheritance diagram for WaitUntilCondition:





## Public Member Functions

- **WaitUntilCondition** ([Condition](#) \*cond)
- bool [run](#) () override

## Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* **withTimeout** (double t\_seconds)
- [AutoCommand](#) \* **withCancelCondition** ([Condition](#) \*true\_to\_end)

## Additional Inherited Members

## Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default\_timeout** = 10.0

### 5.91.1 Detailed Description

Waits until the condition is true.

### 5.91.2 Member Function Documentation

#### 5.91.2.1 [run\(\)](#)

```
bool WaitUntilCondition::run ( ) [inline], [override], [virtual]
```

Executes the command Overridden by child classes

#### Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

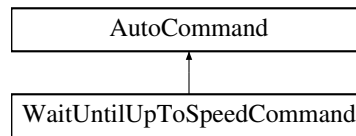
The documentation for this class was generated from the following file:

- include/utils/command\_structure/auto\_command.h

## 5.92 WaitUntilUpToSpeedCommand Class Reference

```
#include <flywheel_commands.h>
```

Inheritance diagram for WaitUntilUpToSpeedCommand:



### Public Member Functions

- [WaitUntilUpToSpeedCommand](#) ([Flywheel](#) &flywheel, int threshold\_rpm)
- bool [run](#) () override

### Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

### Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

#### 5.92.1 Detailed Description

[AutoCommand](#) that listens to the [Flywheel](#) and waits until it is at its target speed +/- the specified threshold

#### 5.92.2 Constructor & Destructor Documentation

##### 5.92.2.1 WaitUntilUpToSpeedCommand()

```
WaitUntilUpToSpeedCommand::WaitUntilUpToSpeedCommand (
    Flywheel & flywheel,
    int threshold_rpm )
```

Creat a [WaitUntilUpToSpeedCommand](#)

## Parameters

<i>flywheel</i>	the flywheel system we are commanding
<i>threshold_rpm</i>	the threshold over and under the flywheel target RPM that we define to be acceptable

## 5.92.3 Member Function Documentation

### 5.92.3.1 run()

```
bool WaitUntilUpToSpeedCommand::run ( ) [override], [virtual]
```

Run spin\_manual Overrides run from [AutoCommand](#)

## Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command\_structure/flywheel\_commands.h
- src/utils/command\_structure/flywheel\_commands.cpp

## 5.93 screen::WidgetConfig Struct Reference

## Public Types

- enum **Type** {  
**Col** , **Row** , **Slider** , **Button** ,  
**Checkbox** , **Label** , **Text** , **Graph** }

## Public Attributes

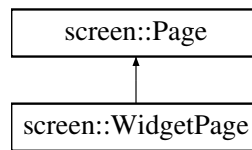
- Type **type**
- union {  
std::vector< [SizedWidget](#) > **widgets**  
[SliderConfig](#) **slider**  
[ButtonConfig](#) **button**  
[CheckboxConfig](#) **checkbox**  
[LabelConfig](#) **label**  
[TextConfig](#) **text**  
[GraphDrawer](#) \* **graph**  
} **config**

The documentation for this struct was generated from the following file:

- include/subsystems/screen.h

## 5.94 screen::WidgetPage Class Reference

Inheritance diagram for screen::WidgetPage:



### Public Member Functions

- **WidgetPage** ([WidgetConfig](#) &cfg)
- void **update** (bool was\_pressed, int x, int y) override  
*collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this [Page](#) (only drawn page gets touch updates))*
- void **draw** (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number) override  
*draw stored data to the screen (runs at 10 hz and only runs if this page is in front)*

### 5.94.1 Member Function Documentation

#### 5.94.1.1 draw()

```

void screen::WidgetPage::draw (
    vex::brain::lcd & screen,
    bool first_draw,
    unsigned int frame_number ) [inline], [override], [virtual]
  
```

draw stored data to the screen (runs at 10 hz and only runs if this page is in front)

#### Parameters

<i>first_draw</i>	true if we just switched to this page
<i>frame_number</i>	frame of drawing we are on (basically an animation tick)

Reimplemented from [screen::Page](#).

#### 5.94.1.2 update()

```

void screen::WidgetPage::update (
    bool was_pressed,
    int x,
    int y ) [override], [virtual]
  
```

collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this [Page](#) (only drawn page gets touch updates))

## Parameters

<i>was_pressed</i>	true if the screen has been pressed
<i>x</i>	x position of screen press (if the screen was pressed)
<i>y</i>	y position of screen press (if the screen was pressed)

Reimplemented from [screen::Page](#).

The documentation for this class was generated from the following file:

- include/subsystems/screen.h

# Chapter 6

## File Documentation

### 6.1 robot\_specs.h

```
00001 #pragma once
00002 #include "../core/include/utils/controls/pid.h"
00003 #include "../core/include/utils/controls/feedback_base.h"
00004
00011 typedef struct
00012 {
00013     double robot_radius;
00014
00015     double odom_wheel_diam;
00016     double odom_gear_ratio;
00017     double dist_between_wheels;
00018
00019     double drive_correction_cutoff;
00020
00021     Feedback *drive_feedback;
00022     Feedback *turn_feedback;
00023     PID::pid_config_t correction_pid;
00024
00025 } robot_specs_t;
```

### 6.2 custom\_encoder.h

```
00001 #pragma once
00002 #include "vex.h"
00003
00008 class CustomEncoder : public vex::encoder
00009 {
00010     typedef vex::encoder super;
00011
00012     public:
00018     CustomEncoder(vex::triport::port &port, double ticks_per_rev);
00019
00025     void setRotation(double val, vex::rotationUnits units);
00026
00032     void setPosition(double val, vex::rotationUnits units);
00033
00039     double rotation(vex::rotationUnits units);
00040
00046     double position(vex::rotationUnits units);
00047
00053     double velocity(vex::velocityUnits units);
00054
00055     private:
00056     double tick_scalar;
00057 };
00058
```

## 6.3 flywheel.h

```

00001 #pragma once
00002
00003 #include "../core/include/utils/controls/feedforward.h"
00004 #include "vex.h"
00005 #include "../core/include/robot_specs.h"
00006 #include "../core/include/utils/controls/pid.h"
00007 #include "../core/include/utils/command_structure/auto_command.h"
00008 #include "../core/include/subsystems/screen.h"
00009 #include <atomic>
00010
00011 class Flywheel
00012 {
00013 public:
00014     // CONSTRUCTORS, GETTERS, AND SETTERS
00015     Flywheel(vex::motor_group &motors, Feedback &feedback, FeedForward &helper, const double ratio,
00016             Filter &filt);
00017
00018     double get_target() const;
00019
00020     double getRPM() const;
00021
00022     vex::motor_group &get_motors() const;
00023
00024     void spin_manual(double speed, directionType dir = fwd);
00025
00026     void spin_rpm(double rpm);
00027
00028     void stop();
00029
00030     bool is_on_target()
00031     {
00032         return fb.is_on_target();
00033     }
00034
00035     screen::Page *Page() const;
00036
00037     AutoCommand *SpinRpmCmd(int rpm)
00038     {
00039         return new FunctionCommand([this, rpm]()
00040                                     { spin_rpm(rpm); return true; });
00041     }
00042
00043     AutoCommand *WaitUntilUpToSpeedCmd()
00044     {
00045         return new WaitUntilCondition(
00046             new FunctionCondition([this]()
00047                                   { return is_on_target(); }));
00048     }
00049 private:
00050     friend class FlywheelPage;
00051     friend int spinRPMTask(void *wheelPointer);
00052
00053     vex::motor_group &motors;
00054     bool task_running = false;
00055     Feedback &fb;
00056     FeedForward &ff;
00057     vex::mutex fb_mut;
00058     double ratio;
00059     std::atomic<double> target_rpm;
00060     task rpm_task;
00061     Filter &avger;
00062
00063     // Functions for internal use only
00064     void set_target(double value);
00065     double measure_RPM();
00066
00067     void spin_raw(double speed, directionType dir = fwd);
00068 };

```

## 6.4 layout.h

```

00001 #include <cmath>
00002 #include <functional>
00003
00004 struct SliderCfg{
00005     double &val;
00006     double min;
00007     double max;

```

```

00008 };
00009
00010
00011

```

## 6.5 lift.h

```

00001 #pragma once
00002
00003 #include "vex.h"
00004 #include "../core/include/utils/controls/pid.h"
00005 #include <iostream>
00006 #include <map>
00007 #include <atomic>
00008 #include <vector>
00009
00010 using namespace vex;
00011 using namespace std;
00012
00020 template <typename T>
00021 class Lift
00022 {
00023     public:
00024
00031     struct lift_cfg_t
00032     {
00033         double up_speed, down_speed;
00034         double softstop_up, softstop_down;
00035
00036         PID::pid_config_t lift_pid_cfg;
00037     };
00038
00060     Lift(motor_group &lift_motors, lift_cfg_t &lift_cfg, map<T, double> &setpoint_map, limit
    *homing_switch=NULL)
00061     : lift_motors(lift_motors), cfg(lift_cfg), lift_pid(cfg.lift_pid_cfg), setpoint_map(setpoint_map),
    homing_switch(homing_switch)
00062     {
00063
00064         is_async = true;
00065         setpoint = 0;
00066
00067         // Create a background task that is constantly updating the lift PID, if requested.
00068         // Set once, and forget.
00069         task t([](void* ptr){
00070             Lift &lift = *((Lift*) ptr);
00071
00072             while(true)
00073             {
00074                 if(lift.get_async())
00075                     lift.hold();
00076
00077                 vexDelay(50);
00078             }
00079
00080             return 0;
00081         }, this);
00082     }
00083
00084
00093     void control_continuous(bool up_ctrl, bool down_ctrl)
00094     {
00095         static timer tmr;
00096
00097         double cur_pos = 0;
00098
00099         // Check if there's a hook for a custom sensor. If not, use the motors.
00100         if(get_sensor == NULL)
00101             cur_pos = lift_motors.position(rev);
00102         else
00103             cur_pos = get_sensor();
00104
00105         if(up_ctrl && cur_pos < cfg.softstop_up)
00106         {
00107             lift_motors.spin(directionType::fwd, cfg.up_speed, volt);
00108             setpoint = cur_pos + .3;
00109
00110             // std::cout << "DEBUG OUT: UP " << setpoint << ", " << tmr.time(sec) << ", " << cfg.down_speed <<
    "\n";
00111
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         "\n";
00121         is_async = true;
00122     } else
00123     {
00124         // Hold the lift at the last setpoint
00125         is_async = true;
00126     }
00127     tmr.reset();
00128 }
00129
00130 void control_manual(bool up_btn, bool down_btn, int volt_up, int volt_down)
00131 {
00132     static bool down_hold = false;
00133     static bool init = true;
00134
00135     // Allow for setting position while still calling this function
00136     if(init || up_btn || down_btn)
00137     {
00138         init = false;
00139         is_async = false;
00140     }
00141
00142     double rev = lift_motors.position(rotationUnits::rev);
00143
00144     if(rev < cfg.softstop_down && down_btn)
00145         down_hold = true;
00146     else if( !down_btn )
00147         down_hold = false;
00148
00149     if(up_btn && rev < cfg.softstop_up)
00150         lift_motors.spin(directionType::fwd, volt_up, voltageUnits::volt);
00151     else if(down_btn && rev > cfg.softstop_down && !down_hold)
00152         lift_motors.spin(directionType::rev, volt_down, voltageUnits::volt);
00153     else
00154         lift_motors.spin(directionType::fwd, 0, voltageUnits::volt);
00155 }
00156
00157 void control_setpoints(bool up_step, bool down_step, vector<T> pos_list)
00158 {
00159     // Make sure inputs are only processed on the rising edge of the button
00160     static bool up_last = up_step, down_last = down_step;
00161
00162     bool up_rising = up_step && !up_last;
00163     bool down_rising = down_step && !down_last;
00164
00165     up_last = up_step;
00166     down_last = down_step;
00167
00168     static int cur_index = 0;
00169
00170     // Avoid an index overflow. Shouldn't happen unless the user changes pos_list between calls.
00171     if(cur_index >= pos_list.size())
00172         cur_index = pos_list.size() - 1;
00173
00174     // Increment or decrement the index of the list, bringing it up or down.
00175     if(up_rising && cur_index < (pos_list.size() - 1))
00176         cur_index++;
00177     else if(down_rising && cur_index > 0)
00178         cur_index--;
00179
00180     // Set the lift to hold the position in the background with the PID loop
00181     set_position(pos_list[cur_index]);
00182     is_async = true;
00183 }
00184
00185 bool set_position(T pos)
00186 {
00187     this->setpoint = setpoint_map[pos];
00188     is_async = true;
00189
00190     return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
00191 }
00192
00193 bool set_setpoint(double val)
00194 {
00195     this->setpoint = val;
00196     return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
00197 }
00198
00199 double get_setpoint()

```

```

00238 {
00239     return this->setpoint;
00240 }
00241
00242 void hold()
00243 {
00244     lift_pid.set_target(setpoint);
00245     // std::cout << "DEBUG OUT: SETPOINT " << setpoint << "\n";
00246
00247     if(get_sensor != NULL)
00248         lift_pid.update(get_sensor());
00249     else
00250         lift_pid.update(lift_motors.position(rev));
00251
00252     // std::cout << "DEBUG OUT: ROTATION " << lift_motors.rotation(rev) << "\n\n";
00253
00254     lift_motors.spin(fwd, lift_pid.get(), volt);
00255 }
00256
00257 void home()
00258 {
00259     static timer tmr;
00260     tmr.reset();
00261
00262     while(tmr.time(sec) < 3)
00263     {
00264         lift_motors.spin(directionType::rev, 6, volt);
00265
00266         if (homing_switch == NULL && lift_motors.current(currentUnits::amp) > 1.5)
00267             break;
00268         else if (homing_switch != NULL && homing_switch->pressing())
00269             break;
00270     }
00271
00272     if(reset_sensor != NULL)
00273         reset_sensor();
00274
00275     lift_motors.resetPosition();
00276     lift_motors.stop();
00277 }
00278
00279 bool get_async()
00280 {
00281     return is_async;
00282 }
00283
00284 void set_async(bool val)
00285 {
00286     this->is_async = val;
00287 }
00288
00289 void set_sensor_function(double (*fn_ptr) (void))
00290 {
00291     this->get_sensor = fn_ptr;
00292 }
00293
00294 void set_sensor_reset(void (*fn_ptr) (void))
00295 {
00296     this->reset_sensor = fn_ptr;
00297 }
00298
00299 private:
00300
00301 motor_group &lift_motors;
00302 lift_cfg_t &cfg;
00303 PID lift_pid;
00304 map<T, double> &setpoint_map;
00305 limit *homing_switch;
00306
00307 atomic<double> setpoint;
00308 atomic<bool> is_async;
00309
00310 double (*get_sensor)(void) = NULL;
00311 void (*reset_sensor)(void) = NULL;
00312
00313 };

```

## 6.6 mecanum\_drive.h

```

00001 #pragma once
00002

```

```

00003 #include "vex.h"
00004 #include "../core/include/utils/controls/pid.h"
00005
00006 #ifndef PI
00007 #define PI 3.141592654
00008 #endif
00009
00014 class MecanumDrive
00015 {
00016
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 };

```

## 6.7 odometry\_3wheel.h

```

00001 #pragma once
00002 #include "../core/include/subsystems/odometry/odometry_base.h"
00003 #include "../core/include/subsystems/tank_drive.h"
00004 #include "../core/include/subsystems/custom_encoder.h"
00005
00032 class Odometry3Wheel : public OdometryBase
00033 {
00034     public:
00035
00040     typedef struct
00041     {
00042         double wheelbase_dist;
00043         double off_axis_center_dist;
00044         double wheel_diam;
00045     } odometry3wheel_cfg_t;
00046
00047
00057     Odometry3Wheel(CustomEncoder &lside_fwd, CustomEncoder &rside_fwd, CustomEncoder &off_axis,
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
offax_delta_deg, pose_t old_pos, odometry3wheel_cfg_t cfg);
00092
00093     CustomEncoder &lside_fwd, &rside_fwd, &off_axis;
00094     odometry3wheel_cfg_t &cfg;
00095
00096
00097 };

```

## 6.8 odometry\_base.h

```

00001 #pragma once
00002
00003 #include "vex.h"
00004 #include "../core/include/utils/geometry.h"
00005 #include "../core/include/robot_specs.h"
00006 #include "../core/include/utils/command_structure/auto_command.h"
00007
00008 #ifndef PI
00009 #define PI 3.141592654
00010 #endif
00011
00012
00013
00026 class OdometryBase
00027 {
00028 public:
00029
00035     OdometryBase(bool is_async);
00036
00041     pose_t get_position(void);
00042
00047     virtual void set_position(const pose_t& newpos=zero_pos);
00048     AutoCommand *SetPositionCmd(const pose_t& newpos=zero_pos);
00053     virtual pose_t update() = 0;
00054
00062     static int background_task(void* ptr);
00063
00069     void end_async();
00070
00077     static double pos_diff(pose_t start_pos, pose_t end_pos);
00078
00085     static double rot_diff(pose_t pos1, pose_t pos2);
00086
00095     static double smallest_angle(double start_deg, double end_deg);
00096
00098     bool end_task = false;
00099
00104     double get_speed();
00105
00110     double get_accel();
00111
00116     double get_angular_speed_deg();
00117
00122     double get_angular_accel_deg();
00123
00127     inline static constexpr pose_t zero_pos = {.x=0.0L, .y=0.0L, .rot=90.0L};
00128
00129 protected:
00133     vex::task *handle;
00134
00138     vex::mutex mut;
00139
00143     pose_t current_pos;
00144
00145     double speed;
00146     double accel;
00147     double ang_speed_deg;
00148     double ang_accel_deg;
00149 };

```

## 6.9 odometry\_tank.h

```

00001 #pragma once
00002
00003 #include "../core/include/subsystems/odometry/odometry_base.h"

```

```

00004 #include "../core/include/subsystems/custom_encoder.h"
00005 #include "../core/include/utils/geometry.h"
00006 #include "../core/include/utils/vector2d.h"
00007 #include "../core/include/utils/moving_average.h"
00008
00009 #include "../core/include/robot_specs.h"
00010
00011 static int background_task(void* odom_obj);
00012
00013
00014
00015
00016
00017
00018
00019
00020 class OdometryTank : public OdometryBase
00021 {
00022 public:
00023     OdometryTank(vex::motor_group &left_side, vex::motor_group &right_side, robot_specs_t &config,
00024                 vex::inertial *imu=NULL, bool is_async=true);
00025
00026     OdometryTank(CustomEncoder &left_custom_enc, CustomEncoder &right_custom_enc, robot_specs_t
00027                 &config, vex::inertial *imu=NULL, bool is_async=true);
00028
00029     OdometryTank(vex::encoder &left_vex_enc, vex::encoder &right_vex_enc, robot_specs_t &config,
00030                 vex::inertial *imu=NULL, bool is_async=true);
00031
00032     pose_t update() override;
00033
00034     void set_position(const pose_t &newpos=zero_pos) override;
00035
00036 private:
00037     static pose_t calculate_new_pos(robot_specs_t &config, pose_t &stored_info, double lside_diff,
00038                                     double rside_diff, double angle_deg);
00039
00040     vex::motor_group *left_side, *right_side;
00041     CustomEncoder *left_custom_enc, *right_custom_enc;
00042     vex::encoder *left_vex_enc, *right_vex_enc;
00043     vex::inertial *imu;
00044     robot_specs_t &config;
00045
00046     double rotation_offset = 0;
00047     ExponentialMovingAverage ema = ExponentialMovingAverage(3);
00048 };

```

## 6.10 screen.h

```

00001 #pragma once
00002 #include "vex.h"
00003 #include <vector>
00004 #include <functional>
00005 #include <map>
00006 #include <cassert>
00007 #include "../core/include/subsystems/odometry/odometry_base.h"
00008 #include "../core/include/utils/graph_drawer.h"
00009 #include "../core/include/utils/controls/pid.h"
00010 #include "../core/include/utils/controls/pidff.h"
00011
00012 namespace screen
00013 {
00014     class ButtonWidget
00015     {
00016     public:
00017         ButtonWidget(std::function<void(void)> onpress, Rect rect, std::string name) :
00018             onpress(onpress), rect(rect), name(name) {}
00019         ButtonWidget(void (*onpress)(), Rect rect, std::string name) : onpress(onpress), rect(rect),
00020             name(name) {}
00021
00022         bool update(bool was_pressed, int x, int y);
00023         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
00024
00025     private:
00026         std::function<void(void)> onpress;
00027         Rect rect;
00028         std::string name = "";
00029         bool was_pressed_last = false;
00030     };
00031
00032     class SliderWidget
00033     {
00034     public:
00035         SliderWidget(double &val, double low, double high, Rect rect, std::string name) : value(val),
00036             low(low), high(high), rect(rect), name(name) {}
00037
00038         bool update(bool was_pressed, int x, int y);

```

```

00064     void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
00065
00066     private:
00067         double &value;
00068
00069         double low;
00070         double high;
00071
00072         Rect rect;
00073         std::string name = "";
00074     };
00075
00076     struct WidgetConfig;
00077
00078     struct SliderConfig
00079     {
00080         double &val;
00081         double low;
00082         double high;
00083     };
00084     struct ButtonConfig
00085     {
00086         std::function<void()> onclick;
00087     };
00088     struct CheckboxConfig
00089     {
00090         std::function<void(bool)> onupdate;
00091     };
00092     struct LabelConfig
00093     {
00094         std::string label;
00095     };
00096
00097     struct TextConfig
00098     {
00099         std::function<std::string()> text;
00100     };
00101     struct SizedWidget
00102     {
00103         int size;
00104         WidgetConfig &widget;
00105     };
00106     struct WidgetConfig
00107     {
00108         enum Type
00109         {
00110             Col,
00111             Row,
00112             Slider,
00113             Button,
00114             Checkbox,
00115             Label,
00116             Text,
00117             Graph,
00118         };
00119         Type type;
00120         union
00121         {
00122             std::vector<SizedWidget> widgets;
00123             SliderConfig slider;
00124             ButtonConfig button;
00125             CheckboxConfig checkbox;
00126             LabelConfig label;
00127             TextConfig text;
00128             GraphDrawer *graph;
00129         } config;
00130     };
00131
00132     class Page;
00133     class Page
00134     {
00135     public:
00136         virtual void update(bool was_pressed, int x, int y);
00137         virtual void draw(vex::brain::lcd &screen, bool first_draw,
00138             unsigned int frame_number);
00139     };
00140
00141     struct ScreenRect
00142     {
00143         uint32_t x1;
00144         uint32_t y1;
00145         uint32_t x2;
00146         uint32_t y2;
00147     };
00148     void draw_widget(WidgetConfig &widget, ScreenRect rect);
00149
00150     class WidgetPage : public Page

```

```

00167     {
00168     public:
00169         WidgetPage(WidgetConfig &cfg) : base_widget(cfg) {}
00170         void update(bool was_pressed, int x, int y) override;
00171
00172         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override
00173         {
00174             draw_widget(base_widget, {.x1 = 20, .y1 = 0, .x2 = 440, .y2 = 240});
00175         }
00176
00177     private:
00178         WidgetConfig &base_widget;
00179     };
00180
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
00200     using draw_func_t = std::function<void(vex::brain::lcd &screen, bool, unsigned int)>;
00201
00202     class StatsPage : public Page
00203     {
00204     public:
00205         StatsPage(std::map<std::string, vex::motor &> motors);
00206         void update(bool was_pressed, int x, int y) override;
00207         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00208
00209     private:
00210         void draw_motor_stats(const std::string &name, vex::motor &mot, unsigned int frame, int x, int
00211 y, vex::brain::lcd &scr);
00212
00213         std::map<std::string, vex::motor &> motors;
00214         static const int y_start = 0;
00215         static const int per_column = 4;
00216         static const int row_height = 20;
00217         static const int row_width = 200;
00218     };
00219
00220     class OdometryPage : public Page
00221     {
00222     public:
00223         OdometryPage(OdometryBase &odom, double robot_width, double robot_height, bool do_trail);
00224         void update(bool was_pressed, int x, int y) override;
00225         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00226
00227     private:
00228         static const int path_len = 40;
00229         static constexpr char const *field_filename = "vex_field_240p.png";
00230
00231         OdometryBase &odom;
00232         double robot_width;
00233         double robot_height;
00234         uint8_t *buf = nullptr;
00235         int buf_size = 0;
00236         pose_t path[path_len];
00237         int path_index = 0;
00238         bool do_trail;
00239         GraphDrawer velocity_graph;
00240     };
00241
00242     class FunctionPage : public Page
00243     {
00244     public:
00245         FunctionPage(update_func_t update_f, draw_func_t draw_t);
00246         void update(bool was_pressed, int x, int y) override;
00247         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00248
00249     private:
00250         update_func_t update_f;
00251         draw_func_t draw_f;
00252     };
00253
00254     class PIDPage : public Page
00255     {
00256     public:
00257         PIDPage(
00258             PID &pid, std::string name, std::function<void(void)> onchange = []() {});
00259         PIDPage(
00260             PIDFF &pidff, std::string name, std::function<void(void)> onchange = []() {});
00261
00262         void update(bool was_pressed, int x, int y) override;

```

```

00290         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00291
00292     private:
00293         void zero_d_f() { cfg.d = 0; }
00294         void zero_i_f() { cfg.i = 0; }
00295
00296         PID::pid_config_t &cfg;
00297         PID &pid;
00298         const std::string name;
00299         std::function<void(void)> onchange;
00300
00301         SliderWidget p_slider;
00302         SliderWidget i_slider;
00303         SliderWidget d_slider;
00304         ButtonWidget zero_i;
00305         ButtonWidget zero_d;
00306
00307         GraphDrawer graph;
00308     };
00309
00310 };
00311
00312 }

```

## 6.11 tank\_drive.h

```

00001 #pragma once
00002
00003 #ifndef PI
00004 #define PI 3.141592654
00005 #endif
00006
00007 #include "vex.h"
00008 #include "../core/include/subsystems/odometry/odometry_tank.h"
00009 #include "../core/include/utils/controls/pid.h"
00010 #include "../core/include/utils/controls/feedback_base.h"
00011 #include "../core/include/robot_specs.h"
00012 #include "../core/include/utils/pure_pursuit.h"
00013 #include "../core/include/utils/command_structure/auto_command.h"
00014 #include <vector>
00015
00016 using namespace vex;
00017
00018 class TankDrive
00019 {
00020 public:
00021     enum class BrakeType
00022     {
00023         None,
00024         ZeroVelocity,
00025         Smart,
00026     };
00027
00028     TankDrive(motor_group &left_motors, motor_group &right_motors, robot_specs_t &config, OdometryBase
00029 *odom = NULL);
00030
00031     AutoCommand *DriveToPointCmd(point_t pt, vex::directionType dir = vex::forward, double max_speed =
00032 1.0, double end_speed = 0.0);
00033     AutoCommand *DriveToPointCmd(Feedback &fb, point_t pt, vex::directionType dir = vex::forward, double
00034 max_speed = 1.0, double end_speed = 0.0);
00035
00036     AutoCommand *DriveForwardCmd(double dist, vex::directionType dir = vex::forward, double max_speed =
00037 1.0, double end_speed = 0.0);
00038     AutoCommand *DriveForwardCmd(Feedback &fb, double dist, vex::directionType dir = vex::forward,
00039 double max_speed = 1.0, double end_speed = 0.0);
00040
00041     AutoCommand *TurnToHeadingCmd(double heading, double max_speed = 1.0, double end_speed = 0.0);
00042     AutoCommand *TurnToHeadingCmd(Feedback &fb, double heading, double max_speed = 1.0, double end_speed
00043 = 0.0);
00044
00045     AutoCommand *TurnDegreesCmd(double degrees, double max_speed = 1.0, double start_speed = 0.0);
00046     AutoCommand *TurnDegreesCmd(Feedback &fb, double degrees, double max_speed = 1.0, double end_speed =
00047 0.0);
00048
00049     AutoCommand *PurePursuitCmd(PurePursuit::Path path, directionType dir, double max_speed = 1, double
00050 end_speed = 0);
00051     AutoCommand *PurePursuitCmd(Feedback &feedback, PurePursuit::Path path, directionType dir, double
00052 max_speed = 1, double end_speed = 0);
00053
00054     void stop();
00055
00056     void drive_tank(double left, double right, int power = 1, BrakeType bt = BrakeType::None);
00057     void drive_tank_raw(double left, double right);
00058
00059     void drive_arcade(double forward_back, double left_right, int power = 1, BrakeType bt =
00060 BrakeType::None);

```



```

00090
00102     bool drive_forward(double inches, directionType dir, Feedback &feedback, double max_speed = 1,
double end_speed = 0);
00103
00113     bool drive_forward(double inches, directionType dir, double max_speed = 1, double end_speed = 0);
00114
00125     bool turn_degrees(double degrees, Feedback &feedback, double max_speed = 1, double end_speed = 0);
00126
00137     bool turn_degrees(double degrees, double max_speed = 1, double end_speed = 0);
00138
00151     bool drive_to_point(double x, double y, vex::directionType dir, Feedback &feedback, double max_speed
= 1, double end_speed = 0);
00152
00165     bool drive_to_point(double x, double y, vex::directionType dir, double max_speed = 1, double
end_speed = 0);
00166
00176     bool turn_to_heading(double heading_deg, Feedback &feedback, double max_speed = 1, double end_speed
= 0);
00185     bool turn_to_heading(double heading_deg, double max_speed = 1, double end_speed = 0);
00186
00190     void reset_auto();
00191
00200     static double modify_inputs(double input, int power = 2);
00201
00214     bool pure_pursuit(PurePursuit::Path path, directionType dir, Feedback &feedback, double max_speed =
1, double end_speed = 0);
00215
00229     bool pure_pursuit(PurePursuit::Path path, directionType dir, double max_speed = 1, double end_speed
= 0);
00230
00231 private:
00232     motor_group &left_motors;
00233     motor_group &right_motors;
00234
00235     PID correction_pid;
00236     Feedback *drive_default_feedback = NULL;
00237     Feedback *turn_default_feedback = NULL;
00238
00239     OdometryBase *odometry;
00240
00241     robot_specs_t &config;
00242
00243     bool func_initialized = false;
00244     bool is_pure_pursuit = false;
00245 };

```

## 6.12 auto\_chooser.h

```

00001 #pragma once
00002 #include "vex.h"
00003 #include <string>
00004 #include <vector>
00005 #include "../core/include/subsystems/screen.h"
00006 #include "../core/include/utils/geometry.h"
00007
00016 class AutoChooser : public screen::Page
00017 {
00018 public:
00024     AutoChooser(std::vector<std::string> paths, size_t def = 0);
00025
00026     void update(bool was_pressed, int x, int y);
00027     void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
00028
00033     size_t get_choice();
00034
00035 protected:
00039     struct entry_t
00040     {
00041         Rect rect;
00042         std::string name;
00043     };
00044
00045     static const size_t width = 380;
00046     static const size_t height = 220;
00047
00048     size_t choice;
00049     std::vector<entry_t> list ;
00050 };

```

## 6.13 auto\_command.h

```

00001
00007 #pragma once
00008
00009 #include "vex.h"
00010 #include <functional>
00011 #include <vector>
00012 #include <queue>
00013 #include <atomic>
00014
00015
00025 class Condition
00026 {
00027 public:
00028     Condition *Or(Condition *b);
00029     Condition *And(Condition *b);
00030     virtual bool test() = 0;
00031 };
00032
00033
00034 class AutoCommand
00035 {
00036 public:
00037     static constexpr double default_timeout = 10.0;
00043     virtual bool run() { return true; }
00047     virtual void on_timeout() {}
00048     AutoCommand *withTimeout(double t_seconds)
00049     {
00050         if (this->timeout_seconds < 0)
00051         {
00052             // should never be timed out
00053             return this;
00054         }
00055         this->timeout_seconds = t_seconds;
00056         return this;
00057     }
00058     AutoCommand *withCancelCondition(Condition *true_to_end) {
00059         this->true_to_end = true_to_end;
00060         return this;
00061     }
00071     double timeout_seconds = default_timeout;
00072     Condition *true_to_end = nullptr;
00073 };
00074
00079 class FunctionCommand : public AutoCommand
00080 {
00081 public:
00082     FunctionCommand(std::function<bool(void)> f) : f(f) {}
00083     bool run()
00084     {
00085         return f();
00086     }
00087
00088 private:
00089     std::function<bool(void)> f;
00090 };
00091
00092 // Times tested 3
00093 // Test 1 -> false
00094 // Test 2 -> false
00095 // Test 3 -> true
00096 // Returns false until the Nth time that it is called
00097 // This is pretty much only good for implementing RepeatUntil
00098 class TimesTestedCondition : public Condition
00099 {
00100 public:
00101     TimesTestedCondition(size_t N) : max(N) {}
00102     bool test() override
00103     {
00104         count++;
00105         if (count >= max)
00106         {
00107             return true;
00108         }
00109         return false;
00110     }
00111
00112 private:
00113     size_t count = 0;
00114     size_t max;
00115 };
00116
00118 class FunctionCondition : public Condition
00119 {
00120 public:
00121     FunctionCondition(

```

```

00122         std::function<bool()> cond, std::function<void(void)> timeout = []() {} ) : cond(cond),
        timeout(timeout)
00123     {
00124     }
00125     bool test() override;
00126
00127 private:
00128     std::function<bool()> cond;
00129     std::function<void(void)> timeout;
00130 };
00131
00132 class IfTimePassed : public Condition
00133 {
00134 public:
00135     IfTimePassed(double time_s);
00136     bool test() override;
00137
00138 private:
00139     double time_s;
00140     vex::timer tmr;
00141 };
00142
00143 class WaitUntilCondition : public AutoCommand
00144 {
00145 public:
00146     WaitUntilCondition(Condition *cond) : cond(cond) {}
00147     bool run() override
00148     {
00149         return cond->test();
00150     }
00151
00152 private:
00153     Condition *cond;
00154 };
00155
00156 class InOrder : public AutoCommand
00157 {
00158 public:
00159     InOrder(const InOrder &other) = default;
00160     InOrder(std::queue<AutoCommand *> cmds);
00161     InOrder(std::initializer_list<AutoCommand *> cmds);
00162     bool run() override;
00163     void on_timeout() override;
00164
00165 private:
00166     AutoCommand *current_command = nullptr;
00167     std::queue<AutoCommand *> cmds;
00168     vex::timer tmr;
00169 };
00170
00171 class Parallel : public AutoCommand
00172 {
00173 public:
00174     Parallel(std::initializer_list<AutoCommand *> cmds);
00175     bool run() override;
00176     void on_timeout() override;
00177
00178 private:
00179     std::vector<AutoCommand *> cmds;
00180     std::vector<vex::task *> runners;
00181 };
00182
00183 class Branch : public AutoCommand
00184 {
00185 public:
00186     Branch(Condition *cond, AutoCommand *false_choice, AutoCommand *true_choice);
00187     ~Branch();
00188     bool run() override;
00189     void on_timeout() override;
00190
00191 private:
00192     AutoCommand *false_choice;
00193     AutoCommand *true_choice;
00194     Condition *cond;
00195     bool choice = false;
00196     bool chosen = false;
00197     vex::timer tmr;
00198 };
00199
00200 class Async : public AutoCommand
00201 {
00202 public:
00203     Async(AutoCommand *cmd) : cmd(cmd) {}
00204     bool run() override;
00205
00206 private:

```

```

00222     AutoCommand *cmd = nullptr;
00223 };
00224
00225 class RepeatUntil : public AutoCommand
00226 {
00227 public:
00231     RepeatUntil(InOrder cmds, size_t repeats);
00235     RepeatUntil(InOrder cmds, Condition *true_to_end);
00236     bool run() override;
00237     void on_timeout() override;
00238
00239 private:
00240     const InOrder cmds;
00241     InOrder *working_cmds;
00242     Condition *cond;
00243 };

```

## 6.14 basic\_command.h

```

00001
00014 #pragma once
00015
00016 #include "../core/include/utils/command_structure/auto_command.h"
00017
00018 //Basic Motor Classes-----
00019
00024 class BasicSpinCommand : public AutoCommand {
00025 public:
00026
00027     //Enumerator for the type of power setting in the motor
00028     enum type {percent,voltage,velocity};
00029
00038     BasicSpinCommand(vex::motor &motor, vex::directionType dir, BasicSpinCommand::type setting,
00039 double power);
00039
00046     bool run() override;
00047
00048 private:
00049
00050     vex::motor &motor;
00051
00052     type setting;
00053
00054     vex::directionType dir;
00055
00056     double power;
00057 };
00062 class BasicStopCommand : public AutoCommand{
00063 public:
00064
00071     BasicStopCommand(vex::motor &motor, vex::brakeType setting);
00072
00079     bool run() override;
00080
00081 private:
00082
00083     vex::motor &motor;
00084
00085     vex::brakeType setting;
00086 };
00087
00088 //Basic Solenoid Commands-----
00089
00094 class BasicSolenoidSet : public AutoCommand{
00095 public:
00096
00103     BasicSolenoidSet(vex::pneumatics &solenoid, bool setting);
00104
00111     bool run() override;
00112
00113 private:
00114
00115     vex::pneumatics &solenoid;
00116
00117     bool setting;
00118 };

```

## 6.15 command\_controller.h

```

00001

```

```

00010 #pragma once
00011 #include <vector>
00012 #include <queue>
00013 #include "../core/include/utils/command_structure/auto_command.h"
00014
00015 class CommandController
00016 {
00017 public:
00018     [[deprecated("Use list constructor instead.")]] CommandController() : command_queue({}) {}
00020
00023     CommandController(std::initializer_list<AutoCommand *> cmds) : command_queue(cmds) {}
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 *> cmds);
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 *> cmds, 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 };

```

## 6.16 delay\_command.h

```

00001
00008 #pragma once
00009
00010 #include "../core/include/utils/command_structure/auto_command.h"
00011
00012 class DelayCommand: public AutoCommand {
00013 public:
00018     DelayCommand(int ms): ms(ms) {}
00019
00025     bool run() override {
00026         vexDelay(ms);
00027         return true;
00028     }
00029
00030 private:
00031     // amount of milliseconds to wait
00032     int ms;
00033 };

```

## 6.17 drive\_commands.h

```

00001
00019 #pragma once
00020
00021 #include "vex.h"
00022 #include "../core/include/utils/geometry.h"
00023 #include "../core/include/utils/command_structure/auto_command.h"
00024 #include "../core/include/subsystems/tank_drive.h"
00025
00026 using namespace vex;
00027
00028
00029 // ==== DRIVING ====
00030
00036 class DriveForwardCommand: public AutoCommand
00037 {
00038 public:
00039     DriveForwardCommand(TankDrive &drive_sys, Feedback &feedback, double inches, directionType dir,
double max_speed=1, double end_speed=0);
00040
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
00080     bool run() override;
00084     void on_timeout() override;
00085
00086 private:
00087     // drive system to run the function on
00088     TankDrive &drive_sys;
00089
00090     // feedback controller to use
00091     Feedback &feedback;
00092
00093     // parameters for turn_degrees
00094     double degrees;
00095     double max_speed;
00096     double end_speed;
00097 };
00098
00099 class DriveToPointCommand: public AutoCommand
00100 {
00101 public:
00102     DriveToPointCommand(TankDrive &drive_sys, Feedback &feedback, double x, double y, directionType
00103 dir, double max_speed = 1, double end_speed = 0);
00104     DriveToPointCommand(TankDrive &drive_sys, Feedback &feedback, point_t point, directionType dir,
00105 double max_speed=1, double end_speed = 0);
00106
00115     bool run() override;
00116
00117 private:
00118     // drive system to run the function on
00119     TankDrive &drive_sys;
00120
00124     void on_timeout() override;
00125
00126     // feedback controller to use
00127     Feedback &feedback;
00128
00129     // parameters for drive_to_point
00130     double x;
00131     double y;
00132     directionType dir;
00133     double max_speed;
00134     double end_speed;
00135 };
00136
00137 class TurnToHeadingCommand: public AutoCommand
00138 {
00139 public:
00140     TurnToHeadingCommand(TankDrive &drive_sys, Feedback &feedback, double heading_deg, double speed =
00141 1, double end_speed = 0);
00142
00154     bool run() override;
00158     void on_timeout() override;
00159
00160 private:
00161     // drive system to run the function on
00162     TankDrive &drive_sys;
00163
00164     // feedback controller to use
00165     Feedback &feedback;
00166
00167     // parameters for turn_to_heading
00168     double heading_deg;
00169     double max_speed;
00170

```

```

00171     double end_speed;
00172 };
00173
00177 class PurePursuitCommand: public AutoCommand
00178 {
00179     public:
00188     PurePursuitCommand(TankDrive &drive_sys, Feedback &feedback, PurePursuit::Path path, directionType
dir, double max_speed=1, double end_speed=0);
00189
00193     bool run() override;
00194
00198     void on_timeout() override;
00199
00200     private:
00201     TankDrive &drive_sys;
00202     PurePursuit::Path path;
00203     directionType dir;
00204     Feedback &feedback;
00205     double max_speed;
00206     double end_speed;
00207
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:
00247     OdomSetPosition(OdometryBase &odom, const pose_t &newpos=OdometryBase::zero_pos);
00248
00254     bool run() override;
00255
00256     private:
00257     // drive system with an odometry config
00258     OdometryBase &odom;
00259     pose_t newpos;
00260 };

```

## 6.18 flywheel\_commands.h

```

00001
00007 #pragma once
00008
00009 #include "../core/include/subsystems/flywheel.h"
00010 #include "../core/include/utils/command_structure/auto_command.h"
00011
00017 class SpinRPMCommand: public AutoCommand {
00018     public:
00024     SpinRPMCommand(Flywheel &flywheel, int rpm);
00025
00031     bool run() override;
00032
00033     private:
00034     // Flywheel instance to run the function on
00035     Flywheel &flywheel;
00036
00037     // parameters for spin_rpm
00038     int rpm;
00039 };
00040
00045 class WaitUntilUpToSpeedCommand: public AutoCommand {
00046     public:
00052     WaitUntilUpToSpeedCommand(Flywheel &flywheel, int threshold_rpm);
00053
00059     bool run() override;
00060
00061     private:
00062     // Flywheel instance to run the function on
00063     Flywheel &flywheel;

```

```

00064
00065     // if the actual speed is equal to the desired speed +/- this value, we are ready to fire
00066     int threshold_rpm;
00067 };
00068
00074 class FlywheelStopCommand: public AutoCommand {
00075     public:
00080     FlywheelStopCommand(Flywheel &flywheel);
00081
00087     bool run() override;
00088
00089     private:
00090     // Flywheel instance to run the function on
00091     Flywheel &flywheel;
00092 };
00093
00099 class FlywheelStopMotorsCommand: public AutoCommand {
00100     public:
00105     FlywheelStopMotorsCommand(Flywheel &flywheel);
00106
00112     bool run() override;
00113
00114     private:
00115     // Flywheel instance to run the function on
00116     Flywheel &flywheel;
00117 };
00118
00124 class FlywheelStopNonTasksCommand: public AutoCommand {
00125     FlywheelStopNonTasksCommand(Flywheel &flywheel);
00126
00132     bool run() override;
00133
00134     private:
00135     // Flywheel instance to run the function on
00136     Flywheel &flywheel;
00137 };

```

## 6.19 bang\_bang.h

```

00001 #include "../core/include/utils/controls/feedback_base.h"
00002
00003 class BangBang : public Feedback
00004 {
00005
00006     public:
00007     BangBang(double threshold, double low, double high);
00016     void init(double start_pt, double set_pt, double start_vel [[maybe_unused]] = 0.0, double end_vel
[[maybe_unused]] = 0.0) override;
00017
00024     double update(double val) override;
00025
00029     double get() override;
00030
00037     void set_limits(double lower, double upper) override;
00038
00042     bool is_on_target() override;
00043
00044     private:
00045     double setpt;
00046     double sensor_val;
00047     double lower_bound, upper_bound;
00048     double last_output;
00049     double threshold;
00050 };

```

## 6.20 feedback\_base.h

```

00001 #pragma once
00002
00010 class Feedback
00011 {
00012     public:
00021     virtual void init(double start_pt, double set_pt, double start_vel = 0.0, double end_vel = 0.0) =
0;
00022
00029     virtual double update(double val) = 0;
00030
00034     virtual double get() = 0;
00035

```



```

00042     virtual void set_limits(double lower, double upper) = 0;
00043
00047     virtual bool is_on_target() = 0;
00048
00049
00050 };

```

## 6.21 feedforward.h

```

00001 #pragma once
00002
00003 #include <math.h>
00004 #include <vector>
00005 #include "../core/include/utils/math_util.h"
00006 #include "../core/include/utils/moving_average.h"
00007 #include "vex.h"
00008
00029 class FeedForward
00030 {
00031     public:
00032
00041     typedef struct
00042     {
00043         double kS;
00044         double kV;
00045         double kA;
00046         double kG;
00047     } ff_config_t;
00048
00049     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);

```

## 6.22 motion\_controller.h

```

00001 #pragma once
00002 #include "../core/include/utils/controls/pid.h"
00003 #include "../core/include/utils/controls/feedforward.h"
00004 #include "../core/include/utils/controls/trapezoid_profile.h"
00005 #include "../core/include/utils/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 };

```

## 6.23 pid.h

```

00001 #pragma once
00002
00003 #include "../core/include/utils/controls/feedback_base.h"
00004 #include "vex.h"
00005 #include <cmath>
00006
00007 using namespace vex;
00008
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
00058     PID(pid_config_t &config);
00059
00072     void init(double start_pt, double set_pt, double start_vel = 0,
00073             double end_vel = 0) override;
00074
00082     double update(double sensor_val) override;
00083
00088     double get_sensor_val() const;
00089
00095     double get() override;
00096
00105     void set_limits(double lower, double upper) override;
00106
00111     bool is_on_target() override;
00112
00116     void reset();
00117
00123     double get_error();
00124
00129     double get_target() const;
00130

```

```

00135 void set_target(double target);
00136
00137 pid_config_t
00138     &config;
00140
00141 private:
00142     double last_error =
00143         0;
00144     double accum_error =
00145         0;
00146
00147     double last_time = 0;
00148     double on_target_last_time =
00149         0;
00150
00151     double lower_limit =
00152         0;
00153     double upper_limit =
00154         0;
00155
00156     double target = 0;
00158     double target_vel = 0;
00160     double sensor_val = 0;
00162     double out = 0;
00165
00166     bool is_checking_on_target =
00167         false;
00168
00169     timer pid_timer;
00172 };

```

## 6.24 pidff.h

```

00001 #pragma once
00002 #include "../core/include/utils/controls/feedback_base.h"
00003 #include "../core/include/utils/controls/feedforward.h"
00004 #include "../core/include/utils/controls/pid.h"
00005
00006 class PIDFF : public Feedback {
00007 public:
00008     PIDFF(PID::pid_config_t &pid_cfg, FeedForward::ff_config_t &ff_cfg);
00009
00018     void init(double start_pt, double set_pt, double start_vel,
00019             double end_vel) override;
00020
00025     void set_target(double set_pt);
00026
00027     double get_target() const;
00028     double get_sensor_val() const;
00036     double update(double val) override;
00037
00046     double update(double val, double vel_setpt, double a_setpt = 0);
00047
00051     double get() override;
00052
00060     void set_limits(double lower, double upper) override;
00061
00065     bool is_on_target() override;
00066
00067     void reset();
00068
00069     PID pid;
00070
00071 private:
00072     FeedForward::ff_config_t &ff_cfg;
00073
00074     FeedForward ff;
00075
00076     double out;
00077     double lower_lim, upper_lim;
00078 };

```

## 6.25 take\_back\_half.h

```

00001 #pragma once
00002 #include "../core/include/utils/controls/feedback_base.h"
00003
00006 class TakeBackHalf : public Feedback
00007 {

```

```

00008
00009 public:
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 };

```

## 6.26 trapezoid\_profile.h

```

00001 #pragma once
00002
00003 const int MAX_TRAPEZOID_PROFILE_SEGMENTS = 4;
00004
00008 typedef struct {
00009     double pos;
00010     double vel;
00011     double accel;
00012
00013 } motion_t;
00014
00019 typedef struct {
00020     double pos_after;
00021     double vel_after;
00022     double accel;
00023     double duration;
00024 } trapezoid_profile_segment_t;
00025
00063 class TrapezoidProfile {
00064 public:
00071     TrapezoidProfile(double max_v, double accel);
00072
00081     motion_t calculate(double time_s, double pos_s);
00082
00089     motion_t calculate_time_based(double time_s);
00090
00097     void set_endpts(double start, double end);
00098
00105     void set_vel_endpts(double start, double end);
00106
00113     void set_accel(double accel);
00114
00121     void set_max_v(double max_v);
00122
00129     double get_movement_time() const;
00130
00131     double get_max_v() const;
00132     double get_accel() const;
00133
00134 private:
00135     double si, sf;
00136     double vi, vf;
00137     double max_v;
00138     double accel;
00139     double duration;
00140
00141     trapezoid_profile_segment_t segments[MAX_TRAPEZOID_PROFILE_SEGMENTS];
00142     int num_acceleration_phases;
00143
00144     bool precalculated;
00145
00151     bool precalculate();
00152
00163     trapezoid_profile_segment_t calculate_kinetic_motion(double si, double vi,

```

```

00164             double v_target);
00165
00173     trapezoid_profile_segment_t calculate_next_segment(double s, double v);
00174 };

```

## 6.27 generic\_auto.h

```

00001 #pragma once
00002
00003 #include <queue>
00004 #include <map>
00005 #include "vex.h"
00006 #include <functional>
00007
00008 typedef std::function<bool(void)> state_ptr;
00009
00014 class GenericAuto
00015 {
00016     public:
00017
00031     [[deprecated("Use CommandController instead.")]]
00032     bool run(bool blocking);
00033
00038     [[deprecated("Use CommandController instead.")]]
00039     void add(state_ptr new_state);
00040
00045     [[deprecated("Use CommandController instead.")]]
00046     void add_async(state_ptr async_state);
00047
00052     [[deprecated("Use CommandController instead.")]]
00053     void add_delay(int ms);
00054
00055     private:
00056
00057     std::queue<state_ptr> state_list;
00058
00059 };

```

## 6.28 geometry.h

```

00001 #pragma once
00002 #include <cmath>
00003
00007 struct point_t
00008 {
00009     double x;
00010     double y;
00011
00017     double dist(const point_t other) const
00018     {
00019         return std::sqrt(std::pow(this->x - other.x, 2) + 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
00072 struct pose_t
00073 {
00074     double x;
00075     double y;
00076     double rot;
00077
00078     point_t get_point()
00079     {
00080         return point_t{.x = x, .y = y};
00081     }
00082 };
00083
00084 struct Rect
00085 {
00086     point_t min;
00087     point_t max;
00088     static Rect from_min_and_size(point_t min, point_t size){
00089         return {min, min+size};
00090     }
00091     point_t dimensions() const
00092     {
00093         return max - min;
00094     }
00095     point_t center() const{
00096         return (min + max)/2;
00097     }
00098     double width() const{
00099         return max.x - min.x;
00100     }
00101     double height() const{
00102         return max.y - min.y;
00103     }
00104     bool contains(point_t p) const
00105     {
00106         bool xin = p.x > min.x && p.x < max.x;
00107         bool yin = p.y > min.y && p.y < max.y;
00108         return xin && yin;
00109     }
00110 };
00111
00112 struct Mat2
00113 {
00114     double X11, X12;
00115     double X21, X22;
00116     point_t operator*(const point_t p) const
00117     {
00118         double outx = p.x * X11 + p.y * X12;
00119         double outy = p.x * X21 + p.y * X22;
00120         return {outx, outy};
00121     }
00122
00123     static Mat2 FromRotationDegrees(double degrees)
00124     {
00125         double rad = degrees * (M_PI / 180.0);
00126         double c = cos(rad);
00127         double s = sin(rad);
00128         return {c, -s, s, c};
00129     }
00130 };

```

## 6.29 graph\_drawer.h

```

00001 #pragma once
00002
00003 #include <string>
00004 #include <stdio.h>
00005 #include <vector>

```

```

00006 #include <cmath>
00007 #include "vex.h"
00008 #include "../core/include/utils/geometry.h"
00009 #include "../core/include/utils/vector2d.h"
00010
00011 class GraphDrawer
00012 {
00013 public:
00020     GraphDrawer(int num_samples, double lower_bound, double upper_bound, std::vector<vex::color> colors,
size_t num_series = 1);
00025     void add_samples(std::vector<point_t> sample);
00026
00031     void add_samples(std::vector<double> sample);
00032
00040     void draw(vex::brain::lcd &screen, int x, int y, int width, int height);
00041
00042 private:
00043     std::vector<std::vector<point_t> series;
00044     int sample_index = 0;
00045     std::vector<vex::color> cols;
00046     vex::color bgcol = vex::transparent;
00047     bool border;
00048     double upper;
00049     double lower;
00050     bool auto_fit = false;
00051 };

```

## 6.30 logger.h

```

00001 #pragma once
00002
00003 #include <cstdint>
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
00042     void Log(const std::string &s);
00043
00047     void Log(LogLevel level, const std::string &s);
00048
00051     void Logln(const std::string &s);
00052
00056     void Logln(LogLevel level, const std::string &s);
00057
00061     void Logf(const char *fmt, ...);
00062
00067     void Logf(LogLevel level, const char *fmt, ...);
00068 };

```

## 6.31 math\_util.h

```

00001 #pragma once
00002 #include <vector>
00003 #include "math.h"

```

```

00004 #include "vex.h"
00005 #include "../core/include/utils/geometry.h"
00006
00007
00015 double clamp(double value, double low, double high);
00016
00023 double lerp(double a, double b, double t);
00030 double sign(double x);
00031
00032 double wrap_angle_deg(double input);
00033 double wrap_angle_rad(double input);
00034
00035 /*
00036 Calculates the variance of a set of numbers (needed for linear regression)
00037 https://en.wikipedia.org/wiki/Variance
00038 @param values the values for which the variance is taken
00039 @param mean the average of values
00040 */
00041 double variance(std::vector<double> const &values, double mean);
00042
00043
00044 /*
00045 Calculates the average of a vector of doubles
00046 @param values the list of values for which the average is taken
00047 */
00048 double mean(std::vector<double> const &values);
00049
00050 /*
00051 Calculates the covariance of a set of points (needed for linear regression)
00052 https://en.wikipedia.org/wiki/Covariance
00053
00054 @param points the points for which the covariance is taken
00055 @param meanx the mean value of all x coordinates in points
00056 @param meany the mean value of all y coordinates in points
00057 */
00058 double covariance(std::vector<std::pair<double, double> const &points, double meanx, double meany);
00059
00060 /*
00061 Calculates the slope and y intercept of the line of best fit for the data
00062 @param points the points for the data
00063 */
00064 std::pair<double, double> calculate_linear_regression(std::vector<std::pair<double, double> const
&points);
00065
00066 double estimate_path_length(const std::vector<point_t> &points);

```

## 6.32 moving\_average.h

```

00001 #pragma once
00002 #include <vector>
00003
00008 class Filter
00009 {
00010 public:
00011     virtual void add_entry(double n) = 0;
00012     virtual double get_value() const = 0;
00013 };
00014
00027 class MovingAverage : public Filter
00028 {
00029 public:
00030     /*
00031      * Create a moving average calculator with 0 as the default value
00032      *
00033      * @param buffer_size The size of the buffer. The number of samples that constitute a valid
reading
00034      */
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
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
00059 double get_value() const override;
00060
00065 int get_size() const;
00066
00067 private:
00068     int buffer_index;           // index of the next value to be overridden
00069     std::vector<double> buffer; // all current data readings we've taken
00070     double current_avg;        // the current value of the data
00071 };
00072
00085 class ExponentialMovingAverage : public Filter
00086 {
00087 public:
00088     /*
00089     * Create a moving average calculator with 0 as the default value
00090     *
00091     * @param buffer_size The size of the buffer. The number of samples that constitute a valid
00092     * reading
00093     */
00093     ExponentialMovingAverage(int buffer_size);
00094     /*
00095     * Create a moving average calculator with a specified default value
00096     * @param buffer_size The size of the buffer. The number of samples that constitute a valid
00097     * reading
00097     * @param starting_value The value that the average will be before any data is added
00098     */
00099     ExponentialMovingAverage(int buffer_size, double starting_value);
00100
00101     /*
00102     * Add a reading to the buffer
00103     * Before:
00104     * [ 1 1 2 2 3 3] => 2
00105     * ^
00106     * After:
00107     * [ 2 1 2 2 3 3] => 2.16
00108     * ^
00109     * @param n the sample that will be added to the moving average.
00110     */
00111 void add_entry(double n) override;
00112
00117 double get_value() const override;
00118
00123 int get_size();
00124
00125 private:
00126     int buffer_index;           // index of the next value to be overridden
00127     std::vector<double> buffer; // all current data readings we've taken
00128     double current_avg;        // the current value of the data
00129 };

```

## 6.33 pure\_pursuit.h

```

00001 #pragma once
00002
00003 #include <vector>
00004 #include "../core/include/utils/geometry.h"
00005 #include "../core/include/utils/vector2d.h"
00006 #include "vex.h"
00007
00008 using namespace vex;
00009
00010 namespace PurePursuit {
00014     class Path
00015     {
00016     public:
00022         Path(std::vector<point_t> points, double radius);
00023
00027         std::vector<point_t> get_points();
00028
00032         double get_radius();
00033
00037         bool is_valid();
00038
00039     private:
00040         std::vector<point_t> points;
00041         double radius;
00042         bool valid;
00043     };

```

```

00048     struct spline
00049     {
00050         double a, b, c, d, x_start, x_end;
00051
00052         double getY(double x) {
00053             return a * pow((x - x_start), 3) + b * pow((x - x_start), 2) + c * (x - x_start) + d;
00054         }
00055     };
00060     struct hermite_point
00061     {
00062         double x;
00063         double y;
00064         double dir;
00065         double mag;
00066
00067         point_t getPoint() const {
00068             return {x, y};
00069         }
00070
00071         Vector2D getTangent() const {
00072             return Vector2D(dir, mag);
00073         }
00074     };
00075
00080     extern std::vector<point_t> line_circle_intersections(point_t center, double r, point_t point1,
point_t point2);
00084     extern point_t get_lookahead(const std::vector<point_t> &path, pose_t robot_loc, double radius);
00085
00089     extern std::vector<point_t> inject_path(const std::vector<point_t> &path, double spacing);
00090
00102     extern std::vector<point_t> smooth_path(const std::vector<point_t> &path, double weight_data, double
weight_smooth, double tolerance);
00103
00104     extern std::vector<point_t> smooth_path_cubic(const std::vector<point_t> &path, double res);
00105
00114     extern std::vector<point_t> smooth_path_hermite(const std::vector<hermite_point> &path, double
step);
00115
00126     extern double estimate_remaining_dist(const std::vector<point_t> &path, pose_t robot_pose, double
radius);
00127
00128 }

```

## 6.34 serializer.h

```

00001 #pragma once
00002 #include <algorithm>
00003 #include <map>
00004 #include <string>
00005 #include <vector>
00006 #include <stdio.h>
00007 #include <vex.h>
00008
00010 const char serialization_separator = '$';
00012 const std::size_t MAX_FILE_SIZE = 4096;
00013
00015 class Serializer
00016 {
00017 private:
00018     bool flush_always;
00019     std::string filename;
00020     std::map<std::string, int> ints;
00021     std::map<std::string, bool> bools;
00022     std::map<std::string, double> doubles;
00023     std::map<std::string, std::string> strings;
00024
00026     bool read_from_disk();
00027
00028 public:
00030     ~Serializer()
00031     {
00032         save_to_disk();
00033         printf("Saving %s\n", filename.c_str());
00034         fflush(stdout);
00035     }
00036
00040     explicit Serializer(const std::string &filename, bool flush_always = true) :
flush_always(flush_always), filename(filename), ints({}), bools({}), doubles({}), strings({})
00041     {
00042         read_from_disk();
00043     }
00044 }
00045

```

```

00047     void save_to_disk() const;
00048
00050
00054     void set_int(const std::string &name, int i);
00055
00059     void set_bool(const std::string &name, bool b);
00060
00064     void set_double(const std::string &name, double d);
00065
00069     void set_string(const std::string &name, std::string str);
00070
00073
00078     int int_or(const std::string &name, int otherwise);
00079
00084     bool bool_or(const std::string &name, bool otherwise);
00085
00090     double double_or(const std::string &name, double otherwise);
00091
00096     std::string string_or(const std::string &name, std::string otherwise);
00097 };

```

## 6.35 vector2d.h

```

00001 #pragma once
00002
00003
00004 #include <cmath>
00005 #include "../core/include/utils/geometry.h"
00006
00007 #ifndef PI
00008 #define PI 3.141592654
00009 #endif
00015 class Vector2D
00016 {
00017 public:
00024     Vector2D(double dir, double mag);
00025
00031     Vector2D(point_t p);
00032
00040     double get_dir() const;
00041
00045     double get_mag() const;
00046
00050     double get_x() const;
00051
00055     double get_y() const;
00056
00061     Vector2D normalize();
00062
00067     point_t point();
00068
00074     Vector2D operator*(const double &x);
00081     Vector2D operator+(const Vector2D &other);
00088     Vector2D operator-(const Vector2D &other);
00089
00090 private:
00091
00092     double dir, mag;
00093
00094 };
00095
00101 double deg2rad(double deg);
00102
00109 double rad2deg(double r);

```

# Index

- accel
  - OdometryBase, 100
- add
  - CommandController, 31, 32
  - GenericAuto, 66
- add\_async
  - GenericAuto, 66
- add\_cancel\_func
  - CommandController, 32
- add\_delay
  - CommandController, 33
  - GenericAuto, 66
- add\_entry
  - ExponentialMovingAverage, 46
  - Filter, 53
  - MovingAverage, 91
- add\_samples
  - GraphDrawer, 68
- AndCondition, 13
  - test, 13
- ang\_accel\_deg
  - OdometryBase, 100
- ang\_speed\_deg
  - OdometryBase, 100
- Async, 14
  - run, 15
- auto\_drive
  - MecanumDrive, 81
- auto\_turn
  - MecanumDrive, 82
- AutoChooser, 15
  - AutoChooser, 16
  - choice, 17
  - draw, 16
  - get\_choice, 16
  - list, 17
  - update, 17
- AutoChooser::entry\_t, 44
  - name, 44
- AutoCommand, 18
  - on\_timeout, 19
  - run, 19
  - timeout\_seconds, 19
- background\_task
  - OdometryBase, 97
- BangBang, 20
  - get, 20
  - init, 20
  - is\_on\_target, 21
  - set\_limits, 21
  - update, 21
- BasicSolenoidSet, 22
  - BasicSolenoidSet, 22
  - run, 23
- BasicSpinCommand, 23
  - BasicSpinCommand, 24
  - run, 25
- BasicStopCommand, 25
  - BasicStopCommand, 26
  - run, 26
- bool\_or
  - Serializer, 133
- BrakeType
  - TankDrive, 145
- Branch, 27
  - on\_timeout, 28
  - run, 28
- ButtonWidget
  - screen::ButtonWidget, 29
- calculate
  - FeedForward, 50
  - TrapezoidProfile, 158
- calculate\_time\_based
  - TrapezoidProfile, 158
- choice
  - AutoChooser, 17
- CommandController, 30
  - add, 31, 32
  - add\_cancel\_func, 32
  - add\_delay, 33
  - CommandController, 31
  - last\_command\_timed\_out, 33
  - run, 33
- Condition, 33
- config
  - PID, 118
- control\_continuous
  - Lift< T >, 73
- control\_manual
  - Lift< T >, 73
- control\_setpoints
  - Lift< T >, 74
- Core, 1
- current\_pos
  - OdometryBase, 100
- CustomEncoder, 34
  - CustomEncoder, 34
  - position, 35

- rotation, 35
  - setPosition, 35
  - setRotation, 36
  - velocity, 36
- DelayCommand, 36
  - DelayCommand, 37
  - run, 37
- dist
  - point\_t, 125
- double\_or
  - Serializer, 133
- draw
  - AutoChooser, 16
  - FlywheelPage, 57
  - GraphDrawer, 68
  - MotionControllerPage, 89
  - screen::FunctionPage, 65
  - screen::OdometryPage, 102
  - screen::Page, 109
  - screen::PIDPage, 124
  - screen::StatsPage, 141
  - screen::WidgetPage, 171
- drive
  - MecanumDrive, 82
- drive\_arcade
  - TankDrive, 146
- drive\_forward
  - TankDrive, 146, 147
- drive\_raw
  - MecanumDrive, 83
- drive\_tank
  - TankDrive, 148
- drive\_tank\_raw
  - TankDrive, 148
- drive\_to\_point
  - TankDrive, 148, 149
- DriveForwardCommand, 38
  - DriveForwardCommand, 39
  - on\_timeout, 39
  - run, 40
- DriveStopCommand, 40
  - DriveStopCommand, 41
  - on\_timeout, 41
  - run, 41
- DriveToPointCommand, 42
  - DriveToPointCommand, 43
  - run, 44
- end\_async
  - OdometryBase, 97
- error\_method
  - PID::pid\_config\_t, 118
- ERROR\_TYPE
  - PID, 114
- ExponentialMovingAverage, 45
  - add\_entry, 46
  - ExponentialMovingAverage, 45
  - get\_size, 46
  - get\_value, 46
- Feedback, 47
  - get, 48
  - init, 48
  - is\_on\_target, 48
  - set\_limits, 48
  - update, 49
- FeedForward, 49
  - calculate, 50
  - FeedForward, 50
- FeedForward::ff\_config\_t, 51
  - kA, 51
  - kG, 51
  - kS, 52
  - kV, 52
- Filter, 52
  - add\_entry, 53
  - get\_value, 53
- Flywheel, 53
  - Flywheel, 54
  - get\_motors, 54
  - get\_target, 54
  - getRPM, 54
  - is\_on\_target, 55
  - Page, 55
  - spin\_manual, 55
  - spin\_rpm, 56
  - SpinRpmCmd, 56
  - spinRPMTask, 57
  - stop, 56
  - WaitUntilUpToSpeedCmd, 56
- FlywheelPage, 57
  - draw, 57
  - update, 57
- FlywheelStopCommand, 58
  - FlywheelStopCommand, 59
  - run, 59
- FlywheelStopMotorsCommand, 59
  - FlywheelStopMotorsCommand, 60
  - run, 60
- FlywheelStopNonTasksCommand, 61
- FunctionCommand, 62
  - run, 63
- FunctionCondition, 63
  - test, 64
- FunctionPage
  - screen::FunctionPage, 64
- GenericAuto, 65
  - add, 66
  - add\_async, 66
  - add\_delay, 66
  - run, 67
- get
  - BangBang, 20
  - Feedback, 48
  - MotionController, 86
  - PID, 115

- PIDFF, 119
- TakeBackHalf, 142
- get\_accel
  - OdometryBase, 97
- get\_angular\_accel\_deg
  - OdometryBase, 97
- get\_angular\_speed\_deg
  - OdometryBase, 98
- get\_async
  - Lift< T >, 74
- get\_choice
  - AutoChooser, 16
- get\_dir
  - Vector2D, 165
- get\_error
  - PID, 115
- get\_mag
  - Vector2D, 165
- get\_motion
  - MotionController, 86
- get\_motors
  - Flywheel, 54
- get\_movement\_time
  - TrapezoidProfile, 159
- get\_points
  - PurePursuit::Path, 113
- get\_position
  - OdometryBase, 98
- get\_radius
  - PurePursuit::Path, 113
- get\_sensor\_val
  - PID, 115
- get\_setpoint
  - Lift< T >, 74
- get\_size
  - ExponentialMovingAverage, 46
  - MovingAverage, 91
- get\_speed
  - OdometryBase, 98
- get\_target
  - Flywheel, 54
  - PID, 115
- get\_value
  - ExponentialMovingAverage, 46
  - Filter, 53
  - MovingAverage, 91
- get\_x
  - Vector2D, 165
- get\_y
  - Vector2D, 165
- getRPM
  - Flywheel, 54
- GraphDrawer, 67
  - add\_samples, 68
  - draw, 68
  - GraphDrawer, 67
- handle
  - OdometryBase, 101
- hold
  - Lift< T >, 74
- home
  - Lift< T >, 74
- IfTimePassed, 69
  - test, 70
- include/robot\_specs.h, 173
- include/subsystems/custom\_encoder.h, 173
- include/subsystems/flywheel.h, 174
- include/subsystems/layout.h, 174
- include/subsystems/lift.h, 175
- include/subsystems/mecanum\_drive.h, 177
- include/subsystems/odometry/odometry\_3wheel.h, 178
- include/subsystems/odometry/odometry\_base.h, 179
- include/subsystems/odometry/odometry\_tank.h, 179
- include/subsystems/screen.h, 180
- include/subsystems/tank\_drive.h, 183
- include/utls/auto\_chooser.h, 184
- include/utls/command\_structure/auto\_command.h, 185
- include/utls/command\_structure/basic\_command.h, 187
- include/utls/command\_structure/command\_controller.h, 187
- include/utls/command\_structure/delay\_command.h, 188
- include/utls/command\_structure/drive\_commands.h, 188
- include/utls/command\_structure/flywheel\_commands.h, 190
- include/utls/controls/bang\_bang.h, 191
- include/utls/controls/feedback\_base.h, 191
- include/utls/controls/feedforward.h, 192
- include/utls/controls/motion\_controller.h, 192
- include/utls/controls/pid.h, 193
- include/utls/controls/pidff.h, 194
- include/utls/controls/take\_back\_half.h, 194
- include/utls/controls/trapezoid\_profile.h, 195
- include/utls/generic\_auto.h, 196
- include/utls/geometry.h, 196
- include/utls/graph\_drawer.h, 197
- include/utls/logger.h, 198
- include/utls/math\_util.h, 198
- include/utls/moving\_average.h, 199
- include/utls/pure\_pursuit.h, 200
- include/utls/serializer.h, 201
- include/utls/vector2d.h, 202
- init
  - BangBang, 20
  - Feedback, 48
  - MotionController, 86
  - PID, 116
  - PIDFF, 119
  - TakeBackHalf, 142
- InOrder, 70
  - on\_timeout, 71
  - run, 71
- int\_or
  - Serializer, 134

- is\_on\_target
  - BangBang, 21
  - Feedback, 48
  - Flywheel, 55
  - MotionController, 87
  - PID, 116
  - PIDFF, 121
  - TakeBackHalf, 143
- is\_valid
  - PurePursuit::Path, 113
- kA
  - FeedForward::ff\_config\_t, 51
- kG
  - FeedForward::ff\_config\_t, 51
- kS
  - FeedForward::ff\_config\_t, 52
- kV
  - FeedForward::ff\_config\_t, 52
- last\_command\_timed\_out
  - CommandController, 33
- Lift
  - Lift< T >, 73
- Lift< T >, 72
  - control\_continuous, 73
  - control\_manual, 73
  - control\_setpoints, 74
  - get\_async, 74
  - get\_setpoint, 74
  - hold, 74
  - home, 74
  - Lift, 73
  - set\_async, 75
  - set\_position, 75
  - set\_sensor\_function, 75
  - set\_sensor\_reset, 76
  - set\_setpoint, 76
- Lift< T >::lift\_cfg\_t, 76
- list
  - AutoChooser, 17
- Log
  - Logger, 78
- Logf
  - Logger, 78
- Logger, 77
  - Log, 78
  - Logf, 78
  - Logger, 77
  - Logln, 79
- Logln
  - Logger, 79
- Mat2, 80
- MecanumDrive, 80
  - auto\_drive, 81
  - auto\_turn, 82
  - drive, 82
  - drive\_raw, 83
  - MecanumDrive, 81
  - MecanumDrive::mecanumdrive\_config\_t, 84
  - modify\_inputs
    - TankDrive, 150
  - motion\_t, 84
  - MotionController, 85
    - get, 86
    - get\_motion, 86
    - init, 86
    - is\_on\_target, 87
    - MotionController, 86
    - set\_limits, 87
    - tune\_feedforward, 87
    - update, 88
  - MotionController::m\_profile\_cfg\_t, 79
  - MotionControllerPage, 88
    - draw, 89
    - update, 89
  - MovingAverage, 89
    - add\_entry, 91
    - get\_size, 91
    - get\_value, 91
    - MovingAverage, 90
  - mut
    - OdometryBase, 101
  - name
    - AutoChooser::entry\_t, 44
  - None
    - TankDrive, 145
  - normalize
    - Vector2D, 166
  - Odometry3Wheel, 92
    - Odometry3Wheel, 93
    - tune, 94
    - update, 94
  - Odometry3Wheel::odometry3wheel\_cfg\_t, 95
    - off\_axis\_center\_dist, 95
    - wheel\_diam, 95
    - wheelbase\_dist, 95
  - OdometryBase, 95
    - accel, 100
    - ang\_accel\_deg, 100
    - ang\_speed\_deg, 100
    - background\_task, 97
    - current\_pos, 100
    - end\_async, 97
    - get\_accel, 97
    - get\_angular\_accel\_deg, 97
    - get\_angular\_speed\_deg, 98
    - get\_position, 98
    - get\_speed, 98
    - handle, 101
    - mut, 101
    - OdometryBase, 97
    - pos\_diff, 98
    - rot\_diff, 99
    - set\_position, 99

- smallest\_angle, 99
  - speed, 101
  - update, 100
  - zero\_pos, 101
- OdometryPage
  - screen::OdometryPage, 102
- OdometryTank, 103
  - OdometryTank, 104, 105
  - set\_position, 106
  - update, 106
- OdomSetPosition, 106
  - OdomSetPosition, 107
  - run, 108
- off\_axis\_center\_dist
  - Odometry3Wheel::odometry3wheel\_cfg\_t, 95
- on\_target\_time
  - PID::pid\_config\_t, 118
- on\_timeout
  - AutoCommand, 19
  - Branch, 28
  - DriveForwardCommand, 39
  - DriveStopCommand, 41
  - InOrder, 71
  - Parallel, 111
  - PurePursuitCommand, 128
  - RepeatUntil, 130
  - TurnDegreesCommand, 161
  - TurnToHeadingCommand, 163
- operator+
  - point\_t, 125
  - Vector2D, 166
- operator-
  - point\_t, 125
  - Vector2D, 167
- operator\*
  - Vector2D, 166
- OrCondition, 108
  - test, 108
- Page
  - Flywheel, 55
- Parallel, 110
  - on\_timeout, 111
  - run, 111
- parallel\_runner\_info, 112
- Path
  - PurePursuit::Path, 112
- PID, 113
  - config, 118
  - ERROR\_TYPE, 114
  - get, 115
  - get\_error, 115
  - get\_sensor\_val, 115
  - get\_target, 115
  - init, 116
  - is\_on\_target, 116
  - PID, 114
  - reset, 116
  - set\_limits, 116
  - set\_target, 117
  - update, 117
- PID::pid\_config\_t, 118
  - error\_method, 118
  - on\_target\_time, 118
- PIDFF, 119
  - get, 119
  - init, 119
  - is\_on\_target, 121
  - set\_limits, 121
  - set\_target, 121
  - update, 122
- PIDPage
  - screen::PIDPage, 123
- point
  - Vector2D, 167
- point\_t, 124
  - dist, 125
  - operator+, 125
  - operator-, 125
- pos\_diff
  - OdometryBase, 98
- pose\_t, 126
- position
  - CustomEncoder, 35
- pure\_pursuit
  - TankDrive, 150, 151
- PurePursuit::hermite\_point, 69
- PurePursuit::Path, 112
  - get\_points, 113
  - get\_radius, 113
  - is\_valid, 113
  - Path, 112
- PurePursuit::spline, 140
- PurePursuitCommand, 126
  - on\_timeout, 128
  - PurePursuitCommand, 127
  - run, 128
- Rect, 128
- RepeatUntil, 129
  - on\_timeout, 130
  - RepeatUntil, 129, 130
  - run, 130
- reset
  - PID, 116
- reset\_auto
  - TankDrive, 152
- robot\_specs\_t, 131
- rot\_diff
  - OdometryBase, 99
- rotation
  - CustomEncoder, 35
- run
  - Async, 15
  - AutoCommand, 19
  - BasicSolenoidSet, 23
  - BasicSpinCommand, 25
  - BasicStopCommand, 26



- Branch, 28
- CommandController, 33
- DelayCommand, 37
- DriveForwardCommand, 40
- DriveStopCommand, 41
- DriveToPointCommand, 44
- FlywheelStopCommand, 59
- FlywheelStopMotorsCommand, 60
- FunctionCommand, 63
- GenericAuto, 67
- InOrder, 71
- OdomSetPosition, 108
- Parallel, 111
- PurePursuitCommand, 128
- RepeatUntil, 130
- SpinRPMCommand, 139
- TurnDegreesCommand, 161
- TurnToHeadingCommand, 163
- WaitUntilCondition, 168
- WaitUntilUpToSpeedCommand, 170

save\_to\_disk

- Serializer, 134

screen::ButtonConfig, 28

screen::ButtonWidget, 28

- ButtonWidget, 29
- update, 30

screen::CheckboxConfig, 30

screen::FunctionPage, 64

- draw, 65
- FunctionPage, 64
- update, 65

screen::LabelConfig, 72

screen::OdometryPage, 101

- draw, 102
- OdometryPage, 102
- update, 102

screen::Page, 109

- draw, 109
- update, 110

screen::PIDPage, 123

- draw, 124
- PIDPage, 123
- update, 124

screen::ScreenData, 131

screen::ScreenRect, 132

screen::SizedWidget, 136

screen::SliderConfig, 136

screen::SliderWidget, 137

- SliderWidget, 137
- update, 137

screen::StatsPage, 140

- draw, 141
- StatsPage, 141
- update, 141

screen::TextConfig, 155

screen::WidgetConfig, 170

screen::WidgetPage, 171

- draw, 171
- update, 171

Serializer, 132

- bool\_or, 133
- double\_or, 133
- int\_or, 134
- save\_to\_disk, 134
- Serializer, 133
- set\_bool, 134
- set\_double, 135
- set\_int, 135
- set\_string, 135
- string\_or, 135

set\_accel

- TrapezoidProfile, 159

set\_async

- Lift< T >, 75

set\_bool

- Serializer, 134

set\_double

- Serializer, 135

set\_endpts

- TrapezoidProfile, 159

set\_int

- Serializer, 135

set\_limits

- BangBang, 21
- Feedback, 48
- MotionController, 87
- PID, 116
- PIDFF, 121
- TakeBackHalf, 143

set\_max\_v

- TrapezoidProfile, 159

set\_position

- Lift< T >, 75
- OdometryBase, 99
- OdometryTank, 106

set\_sensor\_function

- Lift< T >, 75

set\_sensor\_reset

- Lift< T >, 76

set\_setpoint

- Lift< T >, 76

set\_string

- Serializer, 135

set\_target

- PID, 117
- PIDFF, 121

set\_vel\_endpts

- TrapezoidProfile, 160

setPosition

- CustomEncoder, 35

setRotation

- CustomEncoder, 36

SliderCfg, 136

SliderWidget

- screen::SliderWidget, 137

smallest\_angle

- OdometryBase, 99
- Smart
  - TankDrive, 145
- speed
  - OdometryBase, 101
- spin\_manual
  - Flywheel, 55
- spin\_rpm
  - Flywheel, 56
- SpinRpmCmd
  - Flywheel, 56
- SpinRPMCommand, 138
  - run, 139
  - SpinRPMCommand, 139
- spinRPMTask
  - Flywheel, 57
- StatsPage
  - screen::StatsPage, 141
- stop
  - Flywheel, 56
  - TankDrive, 152
- string\_or
  - Serializer, 135
- TakeBackHalf, 142
  - get, 142
  - init, 142
  - is\_on\_target, 143
  - set\_limits, 143
  - update, 143
- TankDrive, 144
  - BrakeType, 145
  - drive\_arcade, 146
  - drive\_forward, 146, 147
  - drive\_tank, 148
  - drive\_tank\_raw, 148
  - drive\_to\_point, 148, 149
  - modify\_inputs, 150
  - None, 145
  - pure\_pursuit, 150, 151
  - reset\_auto, 152
  - Smart, 145
  - stop, 152
  - TankDrive, 145
  - turn\_degrees, 152, 153
  - turn\_to\_heading, 153, 154
  - ZeroVelocity, 145
- test
  - AndCondition, 13
  - FunctionCondition, 64
  - IfTimePassed, 70
  - OrCondition, 108
  - TimesTestedCondition, 156
- timeout\_seconds
  - AutoCommand, 19
- TimesTestedCondition, 155
  - test, 156
- trapezoid\_profile\_segment\_t, 156
- TrapezoidProfile, 156
  - calculate, 158
  - calculate\_time\_based, 158
  - get\_movement\_time, 159
  - set\_accel, 159
  - set\_endpts, 159
  - set\_max\_v, 159
  - set\_vel\_endpts, 160
  - TrapezoidProfile, 158
- tune
  - Odometry3Wheel, 94
- tune\_feedforward
  - MotionController, 87
- turn\_degrees
  - TankDrive, 152, 153
- turn\_to\_heading
  - TankDrive, 153, 154
- TurnDegreesCommand, 160
  - on\_timeout, 161
  - run, 161
  - TurnDegreesCommand, 161
- TurnToHeadingCommand, 162
  - on\_timeout, 163
  - run, 163
  - TurnToHeadingCommand, 163
- update
  - AutoChooser, 17
  - BangBang, 21
  - Feedback, 49
  - FlywheelPage, 57
  - MotionController, 88
  - MotionControllerPage, 89
  - Odometry3Wheel, 94
  - OdometryBase, 100
  - OdometryTank, 106
  - PID, 117
  - PIDFF, 122
  - screen::ButtonWidget, 30
  - screen::FunctionPage, 65
  - screen::OdometryPage, 102
  - screen::Page, 110
  - screen::PIDPage, 124
  - screen::SliderWidget, 137
  - screen::StatsPage, 141
  - screen::WidgetPage, 171
  - TakeBackHalf, 143
- Vector2D, 164
  - get\_dir, 165
  - get\_mag, 165
  - get\_x, 165
  - get\_y, 165
  - normalize, 166
  - operator+, 166
  - operator-, 167
  - operator\*, 166
  - point, 167
  - Vector2D, 164, 165
- velocity

- CustomEncoder, [36](#)
- WaitUntilCondition, [167](#)
  - run, [168](#)
- WaitUntilUpToSpeedCmd
  - Flywheel, [56](#)
- WaitUntilUpToSpeedCommand, [169](#)
  - run, [170](#)
  - WaitUntilUpToSpeedCommand, [169](#)
- wheel\_diam
  - Odometry3Wheel::odometry3wheel\_cfg\_t, [95](#)
- wheelbase\_dist
  - Odometry3Wheel::odometry3wheel\_cfg\_t, [95](#)
- zero\_pos
  - OdometryBase, [101](#)
- ZeroVelocity
  - TankDrive, [145](#)